Proof System of Higher-Order Clausal Logic and its Parallel Implementation(Dissertation_全文)

AUTHOR(S):

Miura, Kinya

Proof System of Higher-Order Clausal Logic and its Parallel Implementation

Kinya MIURA

August 1994

Proof System of Higher-Order Clausal Logic and its Parallel Implementation

Kinya MIURA

Abstract

A logic programming system includes some good features such as pattern matching mechanism or back tracking mechanism. Because of these features, it is employed in the field such as artificial intelligence in which exploratory programming technique is required. There are some merits to extend the logic programming system with higher-order logic because the extension increases its ability by metapredicates, generic procedure and functional programming features. Moreover with the higher-order logic it becomes possible to accomplish an inference about program itself as a metalanguage of the programming language.

On the other hand, however, there arises a problem from the extension. For example,in higher-order logic, logical completeness does not holds. Consequently, a program may not be able to solve a problem in spite of the existence of the solution. Moreover, in an implementation of a higher-order logic programming system, it is necessary to improve its efficiency because of the combinatorial explosion in the execution process. Parallelism is one of measures for it. However, the communication between parallel processes is essential and so we must implement it effective and reliable.

In this thesis, we propose a higher-order logic system named higher-order clausal logic, which is an extension of first order predicate logic, and whose terms and atomic formulae are typed lambda expressions of ω-order.

Thereafter we introduce a semantic framework for the logic. To begin with, we show that proof system with resolution principle has a kind of completeness if we restrict some types of symbols syntactically. And then, we propose a proof system extended with equality axioms, which is less efficient but more powerful enough to be complete in a sense for not restricted higher-order clausal logic. The completeness considered in this thesis is defined on some models in which the ranges of the values of terms of each types are restricted. Under the restriction, any element in the range can be represented as a symbolic expression. Consequently, the range become countable. This kind of completeness is not equal to completeness in original meaning. However, it is significant to consider such a completeness because it is strongly related to correctness in logic programming.

In this thesis, we also consider parallel implementation of the proof system for higher-order clausal logic and we propose a model of processor network and an information disseminating scheme for efficient and reliable broadcasting which is a fundamental operation in the parallel proof system.

This thesis consists of seven chapters. In chapter 1, we describe the background, objectives and motivations. We discuss about features of a logic programming system and consider the problems brought by its higher-order extension especially the problems related to completeness and efficiency of proof system. We also mention some related works and outline of this thesis.

In chapter 2, we propose higher-order clausal logic which we deal with through the thesis. For describing the syntax of the logic, we define types, terms as typed lambda expressions, atomic formulae, clauses and set of clauses. These expressions are regarded as an extension of them in first-order predicate logic with Skolem functions while they include lambda abstraction. We also describe a deduction with resolution principle, which is

the basis of proof systems for higher-order clausal logic.

In chapter 3, we consider model theory for higher-order clausal logic. Generally, model theory for a logic based on typed lambda expressions is specified with a family of domains and meaning functions. Each of the domains corresponds to each of types and no restriction on it are exists originally. In the chapter, we introduce generalized model and its subclass which is derived by restricting the domains, and for which some kinds of completeness can be defined. These notions are prepared for chapter 4 and 5 related to completeness in a logic system.

In chapter 4, we propose a way for extending Herbrand model in first-order predicate logic into higher-order clausal logic as a model class. The extension is difficult in a straightforward manner because of dependencies between atomic formulae. In the chapter, we introduce a restriction on the types of symbols in order to remove the dependencies and we show that proof system based on resolution principle is complete for the model class of extended Herbrand model under the restriction. Under the restriction, higher-order clausal logic still includes first-order predicate logic. Moreover, a term of any type which is constructed solely from 1 is also allowed.

In chapter 5, we define a new model class similar to Herbrand model without any type restriction. We point out that simple proof system based on resolution principle is not complete for the model class. Therefore, we propose an extended proof system with a series of axioms about equality, which is complete for the model class. The proof system is less efficient but more powerful than simple proof system without equality axioms.

In chapter 6, we focus on a parallel implementation of the proof system. In such an implementation, it is significant to execute broadcasting some information such as variable bindings in a efficient and reliable manner.

We propose a model of synchronous processor network and an information disseminating scheme in the chapter, which is nearly time optimal for broadcasting in the case where the number of faulty processor is at most one.

In chapter 7, we summarize the thesis. We also mention unsettled problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Backgrounds

The idea of logic programming was originated in 1974[Kowalski 74] although resolution principle which plays important part in it dated back to 1965[Robinson 65]. Thereafter logic programming systems are used in various fields such as artificial intelligence, deductive database etc. The extensive application of logic programming systems are caused by its some preferable features for these applications. One of the features is an invocation by pattern matching. By the mechanism, processes invoked in the program are decided implicitly depends on the condition in the invocation. Moreover, in such a invocation, variables are not decided as for input or output explicitly. Therefore, it is effective for declarative description of processes.

Backtracking is another preferable feature. This mechanism implies a deterministic implementation of nondeterminism. Therefore, in logic programming systems, nondeterministic programs such as exploratory program

can be described easily.

However these features may be demerits in some cases. It is preferable for a process to be described in a deterministic manner and invoked explicitly if the process always be deterministic and invoked in a fixed manner. Although it is still possible to describe some processes in a deterministic manner on a logic programming system, programmers are often required to be careful.

Comparing with some other programming language, such as LISP, there are more weak points in logic programming systems. One of the weak points is lack of self-descriptiveness. In logic programming systems, program and data are treated as rigorously distinct objects. There is no straight way to operate a program nor a part of a program dissimilar to a generic function in LISP.

A higher-order extension of logic programming will eliminate the demerits in some extent. For example, (typed) lambda calculus as a higher-order extension of logic programming should be regarded as a functional computation mechanism which lets some processes to be described in a deterministic manner and invoked explicitly. Higher-order symbols such as metapredicate or metafunctions are able to operates predicates or functions as a part of a program. Moreover, there is more merit in a higher-order extension of logic programming. In a higher-order logic, mathematical induction can be expressed as a formulae. Therefore a higher-order logic programming system could also be a metaproof system for a program itself. It means that, in such a higher-order logic programming system, both metaprocess such as program verification and programs themselves could be executed in a single and closed system.

On the other hand, however, some problems are arisen from the higher-

order extension. In higher-order logic, logical completeness does not holds. It implies that some programs may not give any solution even in the case where a solution exists certainly. Moreover, in a execution in a higher-order logic programming system, search space is extremely larger than that in a first-order system because of the combinatorial explosion. Therefore, in an implementation of a higher-order logic programming system, it is necessary to improve its efficiency.

There were some related works on higher-order extension of logic programming[Pietrzykowski 72][Miller 86]. In [Pietrzykowski 72], Pietrzykowski and Jensen formalized higher-order unification procedure, and then formalized a proof system based on resolution principle using the procedure. However the proof system was not complete in any sense, although they stated that it was g-complete[1].

Miller and Nadathur formalized and implemented a higher-order programming system[Miller 86]. Their system was excellent in its efficiency. However, completeness of the proof system which the programming system based on was not obvious. They use the term "completeness" not as logical completeness but as equality of procedural semantics and fixed point semantics.

On the other hand, related to efficient implementation of a proof system, parallelism seems to be one of measures for it if the communication between parallel processes such as broadcasting is performed in a efficient and reliable manner. Broadcasting and its fault tolerance on a processor network are significant problems in the field of parallel algorithm[Bienstock 88] [Feige 90][Hedetniemi 88][Liestman 85][Ramanathan 88]. Time optimal

[1]There is a counter example. See example 5.1 in chapter 5.

schemes for broadcasting on hypercubes or binary jumping networks have been known[Fraigniaud 89][Igarashi 90][Johnsson 89] [Han 88]. However these schemes are not identical on their fault tolerance. Even in the case where only one processor was faulty, no time optimal scheme were known.

1.2 Outline of the Thesis

In this thesis, we propose a higher-order logic system named higher-order clausal logic, which is an extension of first order predicate logic, and whose terms and atomic formulae are typed lambda expressions of ω-order. Thereafter we introduce a semantic framework for the logic. To begin with, we show that proof system with resolution principle has a kind of completeness if we restrict some types of symbols syntactically. And then, we propose a proof system extended with equality axioms, which is less efficient but more powerful enough to be complete in a sense for not restricted higher-order clausal logic. The completeness is given by restricting the ranges of the values of terms of each types. The restriction is done in such a way that any element in the range can be represented as a symbolic expression. Consequently, the range become countable.

This kind of completeness is not equal to completeness in original meaning. However, it is significant to consider such a completeness because it is strongly related to correctness in logic programming.

In this thesis, we also consider parallel implementation of a proof system for the higher-order logic, and we propose a model of processor network and an information disseminating scheme for efficient and reliable broadcasting which is a fundamental operation for such a system.

This thesis consists of seven chapters. In chapter 2, we describe the

syntax of higher-order clausal logic which we propose in the thesis. In the beginning, we define a set of types, which consists of a type of truth values, a type of symbols and constructive types. The constructive types are recursively constructed from the two fundamental types. And then we define terms as typed lambda expressions which are constructed from constants and variables with function application and lambda abstraction. Especially terms of the type of truth values are called atomic formulae, from which more complex formulae are constructed. Terms and atomic formulae defined as above are also regarded as higher-order extension of those in first-order predicate logic. Literals, clauses and sets of clauses are defined as in first-order predicate logic. We also describe a deduction and refutation on the higher-order clausal logic with a proof system based on resolution principle after we describe a unification of higher-order terms.

In chapter 3, we consider model theory for higher-order clausal logic. Model theory for a logic based on typed lambda expressions is specified with a family of domains and meaning functions. Each of the domains corresponds to each of types and, originally, no restriction on it are exists.

Generally, any proof system for a higher-order logic could not be complete, that is, there exists a proposition which is true in any models but not provable. Also for higher-order clausal logic, any proof system is not complete in original meaning. In the chapter, we introduce a notions of generalized model and its subclass which is derived by restricting the domains in models. For the kind of models, we define a kind of completeness, which still has practical meaning. These notions are prepared for discussions in chapter 4 and 5.

In chapter 4, we proposed a way for extending Herbrand model in first-order predicate logic into higher-order clausal logic as a model class. The

extension is difficult in a straightforward manner because of dependencies between atomic formulae which is arisen from the fact that an atomic formulae can be appear as a subterm of another atomic formulae. In the chapter, we introduce a restriction on the types of terms in order to remove the dependencies. Under the type restriction, we describe an extension of Herbrand model into higher-order clausal logic, and we show that a proof system based on resolution principle is complete for the model class of extended Herbrand model. In spite of the type restriction, higher-order clausal logic still includes first-order predicate logic. Moreover, terms of any type which is constructed solely from 1 are also allowed.

In chapter 5, we define a new model class similar to Herbrand model without any type restriction. We point out that simple proof system based on resolution principle is not complete for the model class. Therefore, we introduce an extended proof system with a series of axioms about equality, which is complete for the model class. The completeness corresponds to correctness in a logic programming system, that is, any solution is guaranteed to be obtain if the solution can be expressed symbolically. The proof system is less efficient but more powerful than simple proof system without equality axioms.

In chapter 6, we focus on a parallel implementation of the proof system we proposed in the thesis. In the beginning, we summarize the proof procedure and consider parallelism in the proof procedure. And then, we show that, in a parallel implementation of the proof system, it is significant to broadcast some information such as variable bindings in a efficient and reliable manner. Thereafter, we propose a model of synchronous processor network and an information disseminating scheme, which is time optimal for broadcasting if no processor is faulty, and which is time optimal even

if the number of faulty processor is one except some special cases. Even in the special case, the time required for broadcasting exceeds the theoretical lower bound by at most one.

In chapter 7, we summarize the thesis. We also mention unsettled problems.

Chapter 2

Definition of Higher-Order Clausal Logic

2.1 Introduction to Chapter 2

In this chapter, we define syntax of higher-order clausal logic which we propose in the thesis. In the first place, we define a set of types. This set is recursively constructed from two primitive types, the type of truth values and the type of individuals, with a type construction rule which makes a function type from two types. After that, we define typed lambda expressions as terms which are constructed from constants and variables, each of which has a type, with function application and lambda abstraction. Each of terms have types naturally, and terms (term class, strictly) whose type is the type of truth value are defined as atomic formulae. Terms and atomic formulae in this definitions include the terms and atomic formulae in clausal form of first-order predicate logic with Skolem functions. Furthermore, they are more flexible. Literals, clauses and sets of clauses are defined in the sim-

ilar way as in clausal form of first-order predicate logic. Sets of clauses in higher-order clausal logic are, therefore, natural extension of those in the clausal form of first-order predicate logic.

In this chapter, we also give the proof system for higher-order clausal logic which is based on resolution principle. This type of proof system is somewhat simple and popular in logic programming system. In such a proof system, a procedure which unifies two terms is necessary. We describe an outline of an unification procedure for higher-order typed lambda expressions. After that, we describe resolution, deduction and refutation in the proof system.

2.2 Syntax of Higher-Order Clausal Logic

An Atomic formula of higher-order clausal logic is a typed lambda expression. Any atomic formula and its subformulaehave a *type*. The type of atomic formulae is one of primitive types which means truth value, while its subformulae may have other types.

Definition 2.1 (type)

1. $0, 1$ are types. (0 is the type of truth values. 1 is the type of individuals.)

2. If s, t are types, $\langle st \rangle$ is a type. ($\langle st \rangle$ is the type of function which maps an object of type s to an object of type t)

3. Only types which recursively derived from above are types.

□

All function types are unary. multivariable functions should be represented as Curried functions.

Example 2.1

individuals	1
(normal) unary predicates	$\langle 10 \rangle$
(normal) binary functions	$\langle 1\langle 11 \rangle \rangle$
unary predicates whose variable is unary function	$\langle\langle 11 \rangle 0 \rangle$

□

Set of all types are denoted as T in the rest of this thesis.

Depending on these types, we define *vocabulary* which is composed by sets of *constants* and *variables* of each type.

Definition 2.2 (vocabulary) Let C_t be the set of constants of type t, and let V_t be the set of variables of type t. We call the family of C_t and V_t for each type t vocabulary, and express it with $\mathcal{V} = [C_t, V_t]_{t \in T}$. □

We assume that each V_t are countable. For the sake of convenience, we use upper and lower case letters with or without subscription to denote constants and variables respectively. We may also use superscript to express their types explicitly.

On the basis of above definitions, we are able to define *terms* (typed lambda expressions).

Definition 2.3 (term) For arbitrary vocabulary $\mathcal{V} = [C_t, V_t]_{t \in T}$ on a set of types T, terms (on vocabulary $\mathcal{V}$) are defined as the following recursively:

1. An arbitrary element in C_t is a term of type t. (It denotes a *constant.*)

2. An arbitrary element in V_t is a term of type t (It denotes a *variable.*)

3. If $\alpha^{\langle st \rangle}$ is a term of type $\langle st \rangle$, and if β^s is a term of type s, then $(\alpha^{\langle st \rangle}\beta^s)^t$ is a term of type t (It denotes a *function application.*)

4. If x^s is a variable of type s (i.e. $x^s \in V_s$), and if α^t is a term of type t, then $(\lambda x^s.\alpha^t)^{\langle st \rangle}$ is a term of type $\langle st \rangle$. (It denotes a *lambda abstraction.*)

5. Only terms which are recursively derived from above are terms.

□

Superscripts which denote their types may be omitted.

For the sake of convenience, we may use the following abbreviations as usual in lambda calculus:

$$\begin{aligned} (\cdots((Fx_1)x_2)\cdots x_m) &\rightarrow (Fx_1x_2\cdots x_m) \\ (\lambda x_1.(\lambda x_2.(\cdots(\lambda x_n.\alpha)\cdots))) &\rightarrow (\lambda x_1x_2\cdots x_n.\alpha) \end{aligned}$$

and outermost parentheses may be omitted.

Occurrences of variables in terms are often classified in two categories, *bound* and *free*. In this thesis, we use this idea to define term class.

Definition 2.4 (bound, free) An occurrence of variable x is bound in a term α if there is a subterm $\lambda x.\beta$ in the term α and the occurrence of x is in the subterm. Otherwise, Occurrences of variables are free in the term α

□

In lambda calculus, there is some conversions between expressions, which convert expressions to another expressions which have same meanings. These conversions are called α-*conversion* and β-*conversion*. In this thesis, we also must concern these conversions.

Definition 2.5 (α-conversion) Assume that a term α has a subterm $\lambda x.\beta$. Then, $\alpha\{\lambda x'.\beta[x']\}$ is a term derived by α-conversion from α, where $\beta[x']$ is a term derived from β by replacing all free occurrence of x in β to x' and any replaced x' is not bound in $\beta[x']$, and $\alpha\{\lambda x'.\beta[x']\}$ is a term derived from α by replacing the subterm $\lambda x.\beta$ in α to $\lambda x'.\beta[x']$. □

Definition 2.6 (β-conversion) Assume that a term α has a subterm $(\lambda x.\beta)\gamma$. Then, $\alpha\{\beta[\gamma]\}$ is a term derived by β-conversion from α, where $\beta[\gamma]$ is a term derived from β by replacing all free occurrence of x in β to γ and $\alpha\{\beta[\gamma]\}$ is a term derived from α by replacing the subterm $(\lambda x.\beta)\gamma$ in α to $\beta[\gamma]$. □

As meanings of terms should be kept in these conversions, we are able to define an equivalence relation and equivalence class on a set of terms, such that terms which have same meaning are equivalent.

Definition 2.7 (term class) let $\mathcal{T}_\mathcal{V}$ be set of whole terms on a vocabulary $\mathcal{V}$ and let $\overset{\alpha,\beta}{\equiv}$ be a binary relation defined as the following:

1. let $\alpha, \beta \in \mathcal{T}_\mathcal{V}$. Then $\alpha \overset{\alpha,\beta}{\equiv} \beta$ if β is derivable from α by α- or β-conversion.

2. let $\alpha, \beta \in \mathcal{T}_\mathcal{V}$. Then $\alpha \overset{\alpha,\beta}{\equiv} \beta$ if $\beta \overset{\alpha,\beta}{\equiv} \alpha$.

3. let $\alpha, \beta, \gamma \in \mathcal{T}_\mathcal{V}$. Then $\alpha \overset{\alpha,\beta}{\equiv} \gamma$ if $\alpha \overset{\alpha,\beta}{\equiv} \beta$ and $\beta \overset{\alpha,\beta}{\equiv} \gamma$.

Then each elements of the quotient set $\mathcal{T}_\mathcal{V}/\overset{\alpha,\beta}{\equiv}$ are term classes. Each term classes also have their types naturally. □

In each term class, there is a special term which said to be in *normal form*. A term in normal form is a term which cannot be converted by β-conversion. It is well-known that each term classes necessarily have only one normal form in typed lambda expressions.

Any term in normal form of typed lambda expression, The term has the following form:

$$\lambda x_1 \cdots x_m.(\alpha_0 \alpha_1 \cdots \alpha_n)$$

where $x_1 \cdots x_m$ $(m \geq 0)$ are distinct variables, α_0 is a symbol (constant or variable) and $\alpha_1 \cdots \alpha_n$ $(n \geq 0)$ are terms in normal form. According to Huet[Huet 75], we call α_0 the *head* of the term in normal form.

To avoid complication, we may regard a representative term in a term class as the term class itself in the rest of this thesis. We will often elect the normal form as the representative term especially when the term class is operated in a procedure such as a unification.

An *atomic formula* in higher-order clausal logic is a term (class) of type 0. A *literal*, a *clause* and a *set of clauses* in the logic are defined in a similar way as in clausal form of first-order predicate logic.

Definition 2.8 (atomic formula, literal, clause, set of clauses)

1. If α is a term class of type 0, α is an atomic formula.

2. If α is an atomic formula, $+\alpha$,$-\alpha$ are literals. ($+$ and $-$ mean affirmation and negation. $-\alpha$ is called the inverse literal of $+\alpha$, and vice versa.)

3. If $L_1 \cdots L_m$ are literals, a set $\{L_1 \cdots L_m\}$ is a clause. (disjunction of $L_1 \cdots L_m$)

4. If $\mathcal{C}_1 \cdots \mathcal{C}_n$ are clauses, a set $\{\mathcal{C}_1 \cdots \mathcal{C}_n\}$ is a set of clauses. (conjunction of $\mathcal{C}_1 \cdots \mathcal{C}_n$)

□

Empty set of literals is a special clause called empty clause. This clause means contradiction, and so it is important in a reduction to absurdity.

Obviously, a set of clauses in higher-order clausal logic is able to represent any set of clauses in first-order predicate logic with Currying. Furthermore, it has more flexibility and ability to describe various matters. Especially, it is able to treat an atomic formula as a term in another atomic formula. This point is important in two senses. First, it means that this logic is able to treat a proposition as an object and is able to operate it. This feature is important in metaoperation of propositions. Secondly, this feature makes it complicated to construct model theory of the logic. This point should be mentioned at last of chapter 3 more precisely.

Now, we give an example of a set of clauses.

Example 2.2

$\{ \{+P^0\},$
$\{+Q^0\},$
$\{+R^{\langle 00 \rangle} P^0\},$
$\{-R^{\langle 00 \rangle} Q^0\} \}$

This is a set of clauses includes four clauses. Each clause includes only one literal. P and Q are atomic formulae in former two clauses and they are terms in latter two clauses. □

Here is more complicated example.

Example 2.3

$\{ \{+F(\lambda fx.x)(\lambda fx.fx)\},$
$\{+F(\lambda fx.f(pfx))(\lambda fx.qf(p(qf)x)),\ -Fpq\} \}$

This is a set of clauses includes two clauses. The first clause includes one positive literal and the second one includes two literals, positive and negative. □

Because of lambda abstraction, higher-order clausal logic has flexibility and ability to describe various matters. Some more rather complicated examples are shown at the end of chapter5.

2.3 Proof System for Higher-Order Clausal Logic

As in clausal form of first-order predicate logic, we can define a proof system based on resolution principle in higher-order clausal logic. Differences in these proof systems are mainly differences between unification of first-order terms and it of higher-order terms. In this section, we describe unification procedure for higher-order terms and the proof system for higher-order clausal logic based on resolution principle.

2.3.1 Unification

An unification problem for a given set of term classes is to decide whether the set is unifiable or not (decision problem), or to search a most general unifier, mgu of the set (search problem). As far as typed lambda expressions, procedures which find entire mgu's must be semi-algorithm because the number of mgu's may be infinite. It is also known that the decision problem for typed lambda expressions is undecidable[Goldfarb 81]. And so, as far as typed lambda expressions, procedures which solve unification problems must be semi-algorithm.

There are some procedure to solve these problems for typed lambda expressions[Pietrzykowski 72][Huet 75]. For example, Pietrzykowski proposed a non-deterministic semi-algorithm which outputs entire mgu's of any set of term classes[Pietrzykowski 72]. However, this is not so realistic because of its extremely large search space. On the other hand, a procedure which proposed by Huet[Huet 75] is more efficient. Though this procedure is to solve the decision problem, it is easily expanded to find an mgu.

Whichever procedure we adopt, it is not important for completeness of the proof system if the procedure is (semi-)decidable about unifiability (discussed later). So, in this thesis, we adopt the version of Huet[Huet 75] because of its good features.

In this subsection, we will describe an outline of the unification procedure for typed lambda expressions. For more details, please refer Huet [Huet 75].

First of all we define some fundamental notions for preparation. Terms and definitions in this subsection are partly based on Huet[Huet 75].

Definition 2.9 (substitution) A set, each of whose elements is a pair of a term class and a variable whose types are the same, is called substitution if the variables in the pairs are distinct. □

We denote a substitution in the form:

$$\theta = \{x_1 \leftarrow \alpha_1, \cdots, x_n \leftarrow \alpha_n\}$$

where each of x_i $(1 \leq i \leq n)$ is a variable and each of α_i $(1 \leq i \leq n)$ is a term class.

An application of a substitution $\theta = \{x_1 \leftarrow \alpha_1, \cdots, x_n \leftarrow \alpha_n\}$ to an term class β is to derive a class of terms each of which is derived from terms of

term class β by replacing all free occurrence of x_i to α_i. (obviously, derived terms make only one term class.) This definition is naturally extended for a literal, a clause and a set of clauses by applying the substitution to each of term classes which is contained in it. These expressions applied with a substitution are called *instances* of former expressions.

From above definitions the following equations hold obviously:

$$\begin{aligned}(\alpha\theta)\pi &= \alpha(\theta \circ \pi),\\ (\theta \circ \pi) \circ \rho &= \theta \circ (\pi \circ \rho).\end{aligned}$$

If each of term classes α_i $(1 \leq i \leq n)$ in a substitution $\theta = \{x_1 \leftarrow \alpha_1, \cdots, x_n \leftarrow \alpha_n\}$ is a variable which is distinct each other, then the substitution θ is called *renaming substitution*. Obviously, the renaming substitutions, when applied, do not transform any term classes essentially.

Under the above preparation, we can define *unification* and *most general unifier (mgu)*.

Definition 2.10 (unification) Let $\mathcal{T}$ be a set of term classes. Then $\mathcal{T}$ is called unifiable if there is a substitution θ which derives a same term class when applied to each term class in $\mathcal{T}$. A substitution such as θ is called unifier of $\mathcal{T}$,

A unifier is called most general unifier (mgu) of $\mathcal{T}$ if there is no other unifiers π of $\mathcal{T}$ which hold the following conditions:

> There is a substitution ρ which is not renaming substitution and which holds $\theta = \pi \circ \rho$.

□

Obviously, there is at least one mgu for a set of term classes if the set is unifiable. Although the number of mgu's is at most one for any set of terms

in first order predicate logic with constant, it is remarkable that the number of mgu's may be more than one, moreover, it may be infinite (countable).

It is also obvious that, for any unifier π of a set of term classes, there is at least one mgu θ which holds $\theta = \pi \circ \rho$ for some substitution ρ.

Now we describe an outline of a unification procedure. A main objective of the procedure is to construct a tree called a *matching tree*. Each node of the matching tree is labeled with a set of pairs of term classes of a same type. This set is called *disagreement set*, which means that, if each pair of the set labeling a descendant node is unifiable, then the each pair of the set labeling the ancestor node is also unifiable. Each edge of the matching tree is a substitution which reduces the disagreement set of ancestor node to it of descendant node.

The root of the tree is labeled with a disagreement set which contains a pair of term classes which should be unified. If a disagreement set labeling a node contains a pair of term classes which have different constants or bound variables as their heads, the node is a terminal node called failure node because the pair cannot be unified. A node is also a terminal node if all term classes in its disagreement set have free variables as their heads. Such a node called success node because each pair in its disagreement set is easily unifiable. If at least one success node exists in the matching tree, the pair of term classes in the disagreement set of the root is unifiable.

The Huet's procedure expands matching trees from its root. The rules for expanding matching trees are omitted in this thesis because of its complication. Instead of description of these rules, we show an example of a matching tree.

Example 2.4 Figure 2.1 is a example of matching tree. □

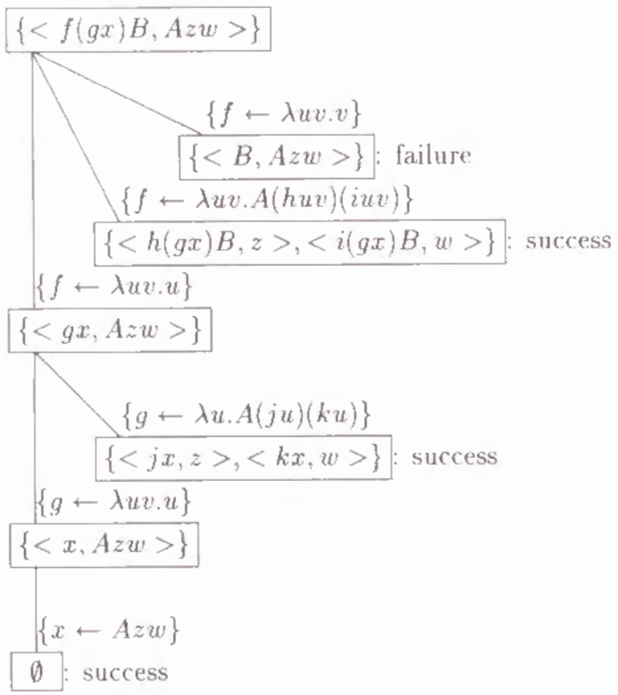


Figure 2.1: A matching tree for a pair of term classes $< f(fx)B, Azw >$

In each success node, we can get a substitution by composing all substitutions on the edges on the path from root to the node. This substitution is not always an mgu but a substitution which is more general than some mgu's. If we need to get exact mgu's we must expand the success nodes which is labeled with non nil disagreement set with some other rules.

Notice that this unification procedure is a kind of tree search, so it is essentially nondeterministic.

2.3.2 Resolution

In this subsection, we describe a inference rule called resolution principle, which is a variation of modus ponens, and which is the only inference rule in our proof system. This inference rule was introduced by Robinson for a proof system for first-order predicate logic[Robinson 65]. In the case of first-order predicate logic, the proof system based on resolution principle is complete without any axiom. On the other hand, in the case of higher-order clausal logic, we may add some axioms or axiom schemata if we need. In fact, although this type of proof system holds a kind of completeness without any axiom in some cases (like in chapter 4), some axioms are needed for completeness in other cases (like in chapter 5).

In this subsection, we describe resolution principle and the proof system for higher-order clausal logic which based on the inference rule without any axiom. An extended proof system with axioms are discussed in chapter 5.

Definition 2.11 (resolution principle) Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two clauses and let $\mathcal{C}_1'$ and $\mathcal{C}_2'$ be subsets of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. If $\mathcal{C}_1'$ and $\mathcal{C}_2'$ include only negative and positive literals respectively and if the set of all atomic formulae in $\mathcal{C}_1'$ and $\mathcal{C}_2'$ are unifiable then the following clause is called *resolvent* of

$\mathcal{C}_1$ and $\mathcal{C}_2$ for any most general unifier θ of the set of atomic formulae.

$$((\mathcal{C}_1 \setminus \mathcal{C}_1') \cup (\mathcal{C}_2 \setminus \mathcal{C}_2'))\theta$$

This inference rule is called *resolution principle* or *resolution* for short. □

Resolution is an inference rule which generates a clause of a resolvent from a set of clauses. Theorem proving is performed by applying this inference rule repeatedly.

Definition 2.12 (deduction, refutation) Let $\mathcal{S}$ be a set of clauses. A sequence of clauses $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n$ is called *deduction* of $\mathcal{C}$ from $\mathcal{S}$ if each of $\mathcal{C}_i$ is a clause in $\mathcal{S}$ or a resolvent of $\mathcal{C}_j$ and $\mathcal{C}_k$ such that $j, k < i$, and if $\mathcal{C}_n = \mathcal{C}$.

A deduction of empty clause from $\mathcal{S}$ is called *refutation* of $\mathcal{S}$. □

When we regard this type of proof system based on resolution principle as a logic programming system, an execution of a program is to get a counter example through a refutation of a set of clauses which denotes the program. And so, to find a refutation of a given set of clauses is essential for our system.

The process of theorem proving in our proof system is nondeterministic because of nondeterminism in the selection of clauses, literals and mgu's in resolution. On the other hand, as see in previous subsection, unification procedure for higher-order term classes is also nondeterministic. These points are not so important if we grasp these procedures as purely nondeterministic procedures. For example, when we discuss about completeness, we can ignore these points. If we consider to implement these procedure as deterministic programs, however, these points arise some problems. These implementation problems are discussed in 6.

Chapter 3

Model Theory for Higher-Order Clausal Logic

3.1 Introduction to Chapter 3

In the last chapter, we have discussed mainly about a proof system for higher-order clausal logic. On the other hand, whenever we discuss about meanings of expressions such as terms, atomic formulae, literal, clause and set of clauses, we must construct another system, called model, which is independent with proof systems and which calculates *values* of expressions in a appropriate way.

An expression is a sequence of symbols and so there are no meanings in itself. A model is a system which gives meanings to symbols in a vocabulary, and which can give a value to each expression uniquely. The way to give meanings is not unique and so there should be a lot of models for a vocabulary. Some expressions, however, are evaluates as *true* by every model. Such expressions are called *valid*. In contrast, expressions which are

evaluated as *false* by every model are called *unsatisfiable*.

Validity and unsatisfiability in terms of model system are strongly related with provability and refutability in terms of proof system respectively. Completeness of a proof system means that any valid expression can be provable or, in a reduction to absurdity, any unsatisfiable expression can be refutable.

When we regard our proof system as a programming system, a program with input data, an execution and a solution are correspond to a set of clauses, a refutation and a counter example respectively. From this view point, completeness of the proof system guarantees that if a program with input data (i.e. a set of clauses) has some valid solutions (i.e. counter examples) then there exists an execution of the program (i.e. a refutation of the set of clauses), which is performed in finite time. Therefore completeness of a proof system corresponds to correctness of a program. This is why we make a study of completeness of our proof system in this thesis. However any proof system for higher-order logic cannot be complete generally. To overcome this situation, we introduce extended notion of completeness, which is based on extended model theory, and which is meaningful in a proof system as a programming system.

In this chapter, we introduce model theory of higher-order clausal logic. In the first section, we introduce standard model and generalized model, which is an extension of standard model and which was introduced by Henkin in [Henkin 50]. We also introduce model class, which is subclass of generalized model. In the second section, we discuss about completeness of higher-order clausal logic and we introduce an extended notion of completeness.

3.2 Generalized Model

A model consists of a family of domains and meaning functions indexed by types. Each domain indexed by a type is a set which consists of values as which constants and variables of the type can be evaluated. A family of domains are called *frame*. A meaning function indexed by a type maps constants of the type to values in the domain of the type.

Definition 3.1 (frame) Let T be a entire set of types. A frame is a family $[D_t]_{t\in T}$ which holds:

$$D_0 = \{0, 1\}, \quad \text{(truth values)}$$
$$D_1 = D, \quad \text{(an arbitrary set of individuals)}$$
$$D_{(st)} = \{f | f : D_s \longrightarrow D_t\}, \quad \text{(functions from } D_s \text{ to } D_t\text{)}$$

□

A model consists of a frame and meaning functions.

Definition 3.2 (model) Let T be a entire set of types and let $\mathcal{V} = [C_t, V_t]_{t\in T}$ be a vocabulary. A model on a vocabulary $\mathcal{V}$ is a family $[D_t, m_t]_{t\in T}$ where $[D_t]_{t\in T}$ is a frame and each m_t is a function from C_t to D_t. □

To evaluate variables, other functions are necessary.

Definition 3.3 (assignment) Let T be a entire set of types and let $\mathcal{V} = [C_t, V_t]_{t\in T}$ be a vocabulary. An assignment on a vocabulary $\mathcal{V}$ is a family $\mathbf{a} = [\mathrm{a}_t]_{t\in T}$ where each a_t is a function from V_t to D_t. □

Now, values of expressions are calculated depending on a model (and an assignment).

Definition 3.4 (value function) Let $M = [D_t, m_t]_{t\in T}$ be a model on a vocabulary $\mathcal{V} = [C_t, V_t]_{t\in T}$ and let $[\mathrm{a}_t]_{t\in T}$ be an assignment on $\mathcal{V}$. A value function $V_{M\mathbf{a}}$ is defined recursively as below:

1. $\forall A \in C_t$, $V_{M\mathbf{a}}[A] = m_t[A]$.

2. $\forall x \in V_t$, $V_{M\mathbf{a}}[x] = \mathrm{a}_t[x]$.

3. Let α and β be terms. Then:

$$V_{M\mathbf{a}}[(\alpha\beta)] = (V_{M\mathbf{a}}[\alpha])[V_{M\mathbf{a}}[\beta]]$$

4. Let x be a variable of type s, let α be a term of type t and let $\mathbf{a}\{d/x\}$ be an assignment which is equal to $\mathbf{a}$ except it assigns d for x. Then $V_{M\mathbf{a}}[(\lambda x.\alpha)]$ is a function f such that:

$$f[d] = V_{M\mathbf{a}\{d/x\}}[\alpha]$$

for all d in D_s.

5. Let α be an atomic formula. Then:

$$V_{M\mathbf{a}}[+\alpha] = V_{M\mathbf{a}}[\alpha]$$
$$V_{M\mathbf{a}}[-\alpha] = 1 - V_{M\mathbf{a}}[\alpha]$$

(Although α is not a term but a term class really, all terms in the term class is apparently mapped to same value by $V_{M\mathbf{a}}$. And so, we regard the value as the value of the term class.)

6. Let $L_1, \cdots, L_m$ be literals. Then:

$$V_{M\mathbf{a}}[\{L_1, \cdots, L_m\}] = \min_{\mathbf{a}'}\{\max_{j=1}^{m}\{V_{M\mathbf{a}'}[L_j]\}\}$$

where $\mathbf{a}'$ moves entire assignments on $\mathcal{V}$.

7. Let $\mathcal{C}_1, \cdots, \mathcal{C}_n$ be clauses. Then:

$$V_{M\mathbf{a}}[\{\mathcal{C}_1, \cdots, \mathcal{C}_n\}] = \min_{i=1}^{n}\{V_{M\mathbf{a}}[\mathcal{C}_i]\}$$

□

In the last two cases in the above definition, assignment $\mathbf{a}$ does not play any role. In these cases, we may abbreviate $V_{M\mathbf{a}}$ as V_M.

In the next section, we discuss about an extended version of completeness of higher-order clausal logic. As preparations for that, in this section, we introduce generalized model(g-model). This idea was introduced by Henkin in [Henkin 50]. G-models include entire standard models and so g-model versions of validity and unsatisfiability become more strong notions.

The major difference between standard model and g-model is their domains. In g-model, domains may be a subset. A family of such domains are called g-frame.

Definition 3.5 (g-frame) Let T be a entire set of types. A g-frame is a family $[D_t]_{t\in T}$ which holds:

$$D_0 = \{0, 1\}, \quad \text{(truth values)}$$
$$D_1 = D, \quad \text{(an arbitrary set of individuals)}$$
$$D_{(st)} \subset \{f | f : D_s \longrightarrow D_t\}, \quad \text{(functions from } D_s \text{ to } D_t\text{)}$$

□

A g-model is a model on g-frame.

Definition 3.6 (g-model) Let T be a entire set of types and let $\mathcal{V} = [C_t, V_t]_{t\in T}$ be a vocabulary. A g-model on a vocabulary $\mathcal{V}$ is a family $[D_t, m_t]_{t\in T}$ where $[D_t]_{t\in T}$ is a g-frame, each m_t is a function from C_t to D_t and value function (see Definition 3.4) can be defined properly. □

The last condition about value function is added to avoid the situation in which f in 4. of definition 3.4 becomes out of $D_{(st)}$.

We call subclass of g-models a *model class*.

Definition 3.7 (model class) Subclass of g-models are called model class. □

Obviously, a class of standard models are a model class.

3.3 Completeness in Higher-Order Clausal Logic

Before discussing about completeness, we define some notions.

Definition 3.8 (g-valid, g-unsatisfiable) A set of clauses $\mathcal{S}$ is called g-valid if and only if $V_M[S] = 1$ for any g-model M. A set of clauses $\mathcal{S}$ is called g-unsatisfiable if and only if $V_M[\mathcal{S}] = 0$ for any g-model M. □

In this thesis, we only consider proof systems which based on refutation. So we define g-completeness like below.

Definition 3.9 (g-complete) A proof system which is able to refute any g-unsatisfiable set of clauses is called g-complete. □

These notions are easily extend to the notions on a model class by reading g-model as model in a model class. As mentioned before, standard model is a model class and so completeness of original meaning is equal to completeness for the model class.

If we consider standard frames whose D_1 is countable set, almost domains in the frames become uncountable sets. So, in these uncountable

domains, there exist elements which cannot be expressed as terms on some models. Hence any proof system which refute a given set of clauses by searching counter examples cannot refute a set of clauses on a vocabulary even if the set of clauses is unsatisfiable and if the counter examples cannot be expressed in the vocabulary. In other words, such a proof system, including our proof system, may not be complete in original meaning[1].

When we regard a proof system as a programming system, however, we are only interested in solutions, that is, counter example which can be expressed in a vocabulary. So it seems adequate to consider only (g-)models whose domains are consists of only elements which can be expressed in the vocabulary when we consider completeness of a proof system as a programming system.

G-models of this kind are expected to construct a model class. Indeed, in first-order predicate logic, Herbrand model is such a model class. If we can construct such a model class for higher-order clausal logic, it seems adequate that we regard that a proof system which is complete on the model class is a correct programming system.

In first-order predicate logic, a set of all atomic formulae which do not include any variable is called Herbrand base. And then, Herbrand models in first-order predicate logic are specified by a set of literals which are obtained by adding plus or minus sign to every element of Herbrand base and in which all literals are regarded to be true.

In higher-order clausal logic, a set of atomic formulae which corresponds to Herbrand base seems to be a set of closed atomic formulae, that is atomic formulae in which any variable does not occur freely. Every set of literals

[1]Indeed, any proof system does not complete.[Gödel 31]

constructed from the set of atomic formulae, however, does not always correspond to a g-model because values of atomic formulae in the set are not independent each other. For example, the set I in example 3.1 does not correspond to any g-model, because it is a contradiction that P and Q have a same value, true, on the other hand (RP) and (RQ) have different values.

Example 3.1 Let $\mathcal{V} = [C_t, V_t]_{t \in T}$ be a vocabulary, let P and Q be in C_0 and let R be in $C_{\langle 00 \rangle}$. I is a set described as the following equation:

$$I = \{+\alpha \mid \alpha \text{ is a closed atomic formula except } (RQ)\} \cup \{-(RQ)\}$$

And so I includes $+P$, $+Q$, $+(RP)$ and $-(RQ)$. □

This contradiction is caused by the fact that closed atomic formulae is distinguishable each other as subterms of other atomic formulae although the values of the atomic formulae must be one of the two values, 1 or 0.

So, to construct a model class which corresponds to Herbrand model in higher-order clausal logic, some more idea is necessary. One idea is to avoid appearances of atomic formulae in other atomic formulae. This idea is achieved by restricting the types of symbols in the vocabulary. This approach is discussed in chapter 4

Another idea is to consider not the set of closed atomic formulae but the quotient set of the set by a kind of equality which is derived from equality in atomic formulae. This idea is discussed in chapter 5.

Chapter 4

Higher-Order Clausal Logic with Type Restriction

4.1 Introduction to Chapter 4

As discussed in the last chapter, it is not so easy to extend Herbrand model in higher-order clausal logic because of dependencies between atomic formulae in the extended Herbrand base, that is, a set of closed atomic formulae. This dependencies are caused the fact that an atomic formula is able to include other atomic formulae. Hence this dependencies can be removed if an atomic formula never include any other atomic formulae.

In this chapter, we introduce a syntactic constraint — type restriction in a vocabulary, and we show that an atomic formula never include any other atomic formulae under the constraint. We also construct a higher-order extension of Herbrand model as a model class and prove that the proof system for higher-order clausal logic based on resolution principle is complete for the model class.

4.2 A Higher-Order Extension of Herbrand Model with Type Restriction

In first order predicate logic, Herbrand universe and Herbrand base are defined as the set of entire terms and the set of atomic formulae which include no variable, respectively.

In higher-order clausal logic, terms are able to have many types, not only 0 and 1, and so we must consider for each types. It seems to natural that the each set corresponding to Herbrand universes is defined as the set of entire closed terms[1] of each type because bound variables in a term are literally bound. We call the each set *H-universe*.

In above case, Herbrand base corresponds to the set of type 0. We call it *H-base*.

Definition 4.1 (H-universe, H-base) Let t be a type. Then H_t denotes entire set of the terms of type t on a vocabulary in which no variables occur freely. H_t called H-universe of type t, and H_0 called H-base especially. □

In first order predicate logic, Herbrand model is defined on the domain of individuals which is isomorphic with Herbrand universe. Hence, in higher-order clausal logic, it is natural that an extension of Herbrand model is defined on a g-frame which is isomorphic with H-universes except the domain of type 0. We call it *H-model*.

Definition 4.2 (H-model) Let T' be the set of entire types of subterms in atomic formulae, let T'' be the set of entire types of subterms which

[1] term classes, exactly

appear only as functions (left hand side of function application[2]) in atomic formulae (obviously, $T \supset T' \supset T''$) and let $M = [D_t, m_t]_{t \in T}$ be a g-model. If there exists a family of mappings $[\Phi_t^M]_{t \in T'}$ which satisfies:

1. Each Φ_t^M is a mapping from H_t to D_t.
2. For any $t \in T' \setminus T'' \setminus \{0\}$, Φ_t^M is bijective.
3. Let H_t' be the set:

   $$\{\alpha \mid \alpha \in H_t \text{ and } \alpha \text{ is not a lambda abstraction}\}$$

   and let $\Phi_t^{M'}$ be the partial function of Φ_t^M whose domain is restricted to H_t'. For any $t \in T''$, $\Phi_t^{M'}$ is bijective. (Note that if $t \in T''$, lambda abstractions of type t never appear as subterms in atomic formulae.)
4. For any $A \in C_t$ such that $t \in T'$, $\Phi_t^M[A] = m_t[A]$.

then M is called H-model.

Obviously, an H-model is also g-model, so all H-models form a model class. □

In this definition, we do not consider terms which cannot appear in any atomic formula because evaluation of such terms plays no part when we evaluate atomic formulae.

Note that each of Φ_t^M is a partial function of a value function. Hence, except $t = 0$, values of any two different closed subterms of atomic formulae are different each other and any element which is not a value of any closed term does not exist because Φ_t^M is a bijection except $t = 0$.

[2] In this definition we consider only terms of normal form — term classes more exactly. So functions cannot be lambda-abstraction.

Generally, Φ_0^M is not injective. Therefore, if T' include 0, proper Φ_t^M's do not exist because of 2 in definition 4.2. This is another aspect of the problem that was mentioned at the end of chapter 3.

To avoid the situation $0 \in T'$, we propose type restriction on a vocabulary, which is performed by forbidding any symbols (constants and variables) of some types in a vocabulary. A trivial solution is to forbid all symbols of the types formed as $\langle s0 \rangle$ for any type s. But this solution is not practical at all[3]. Another solution is to forbid all symbols of the types formed as $\langle s_1 \langle \cdots \langle s_{i-1} \langle r \langle s_{i+1} \langle \cdots \langle s_n s_{n+1} \rangle \cdots \rangle \rangle \rangle \rangle \cdots \rangle \rangle$ $(1 \leq i \leq n)$ for any types $s_1, \cdots, s_{n+1}, t_1, \cdots, t_m$ and r which is formed as $r = \langle t_1 \langle \cdots \langle t_m 0 \rangle \cdots \rangle \rangle$ $(m \geq 0)$. Under this restriction, no atomic formula is not able to appear in another atomic formula. It can be proved.

Theorem 4.1 Let r be a type formed as $\langle t_1 \langle \cdots \langle t_m 0 \rangle \cdots \rangle \rangle$ and let q be a type formed as $\langle s_1 \langle \cdots \langle s_{i-1} \langle r \langle s_{i+1} \langle \cdots \langle s_n s_{n+1} \rangle \cdots \rangle \rangle \rangle \rangle \cdots \rangle \rangle$ for any integer i, n, m such that $1 \leq i \leq n$ and $m \geq 0$ and for any types $s_1, \cdots, s_{n+1}, t_1, \cdots, t_m$. If, for any q, $C_q = V_q = \emptyset$ in a vocabulary $[C_t, V_t]_{t \in T}$, any atomic formula does not contain any other atomic formulae.

Proof. Assume an atomic formulae δ has a proper subterm α of type 0. Because 0 is a primitive type and we only consider terms of normal form, δ has a subterm β which is formed as $(F\gamma_1 \cdots \gamma_{i-1}(\lambda x_1 \cdots x_m.\alpha)\gamma_{i+1} \cdots \gamma_n)$ where $1 \leq i \leq n$, $m \geq 0$, F is a symbol (a constant or a variable), each γ_i is a term and each x_j is a variable[4]. Then the term $(\lambda x_1 \cdots x_m.\alpha)$ has a type r formed as $\langle t_1 \langle \cdots \langle t_m 0 \rangle \cdots \rangle \rangle$. So symbol F has a type q formed as $\langle s_1 \langle \cdots \langle s_{i-1} \langle r \langle s_{i+1} \langle \cdots \langle s_n s_{n+1} \rangle \cdots \rangle \rangle \rangle \rangle \cdots \rangle \rangle$, that is, there is at least one

[3]Indeed, this restriction reduces higher-order clausal logic to propositional logic.

[4]If $m = 0$, $(\lambda x_1 \cdots x_m.\alpha)$ is regarded as α.

symbol of type q in considering vocabulary. Hence, if there is no such symbol in the considering vocabulary, any atomic formula δ does not include any proper subterm α of type 0. □

This theorem means a sufficient condition. So there may be other type restrictions by which we can avoid appearance of atomic formulae in an atomic formula. Our restriction, however, has good properties. Primarily, any first-order term is allowed even under the restriction. Moreover, any symbol of the types which is constructed not from type 0 but from type 1 is also allowed.

The following corollary are immediate from theorem 4.1

Corollary 4.1 Let I be the set of literals which is obtained by adding a sign, plus or minus, to each element in H-base H_0, that is $I = \{s_1 A_1, s_2 A_2, \cdots\}$ for $H_0 = \{A_1, A_2, \cdots\}$ and each $s_1, s_2, \cdots$ is $+$ or $-$. Then, under the restriction of vocabulary in theorem 4.1, there exists a H-model which satisfies all elements in I.

Proof. From theorem 4.1, $0 \notin T'$ in definition 4.2. So proper $[\Phi_t^M]_{t \in T}$ in any H-model M is definable and Φ_0^M is arbitrary definable. Hence truth value of each element in H-base are arbitrary decidable. Immediately, it is possible to construct a H-model which satisfies a any element in given I. □

We call the set of literals such as I in corollary 4.1 *H-Interpretation.*

Definition 4.3 (H-interpretation) Let H_0 is a H-base. Then, a set of literals I is called a (total) H-interpretation if it contains $+A$ or $-A$ exclusively for any A in H_0 and if it contains no other literals. A subset of a total H-interpretation is called partial H-interpretation.

For a ground literal L and partial or total H-interpretation I, L is called to be satisfied by I if L is included in I, and L is called to be falsified by I if the inverse literal of L is included in I. For a ground clause $\mathcal{C} = \{L_1, \cdots, L_m\}$, $\mathcal{C}$ is called to be satisfied by I if at least one of L_i's is satisfied by I, and $\mathcal{C}$ is called to be falsified by I if all of L_i's are satisfied by I. For a ground set of clause $\mathcal{S} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$, $\mathcal{S}$ is called to be satisfied by I if all of $\mathcal{C}_i$'s are satisfied by I, and $\mathcal{S}$ is called to be falsified by I if at least one of $\mathcal{C}_i$'s is satisfied by I. □

4.3 Completeness for the Higher-Order Extension of Herbrand Model

As see in the last section. Under the type restriction proposed in section 4.2, H-model can be defined. In this section, we also show that the proof system which described in chapter 2 is complete for the model class of H-models. This proof of the completeness is similar to that on first-order predicate logic, using a semantic tree.

First of all we give some definitions about a complete semantic tree. Then we prove the theorem about the completeness after proofs of a theorem and a lemma.

Definition 4.4 (complete semantic tree) Let H_0 be a H-base. A complete semantic tree is a binary tree B which satisfies the following conditions:

1. Each edge of B is labeled with a literal which is made from an element of H_0.

2. A pair of literals labeling edges descend from one node is a dual pair, that is, both literals are made from a same element of H_0 and their signs are different each other.

3. For each node N in B, all literals which are labeling edges in the path from the root to N are made from distinct element of H_0.

4. For any element α of H_0, and any downward path P from the root to a leaf[5], there exists an edge in P which is labeled with $+\alpha$ or $-\alpha$.

□

We show an example of complete semantic tree in figure 4.1

Each downward path from the root to a leaf in a complete semantic tree corresponds to an H-interpretation, which includes all literals labeling the

[5]If H_0 is infinite, the length of the path is also infinite.

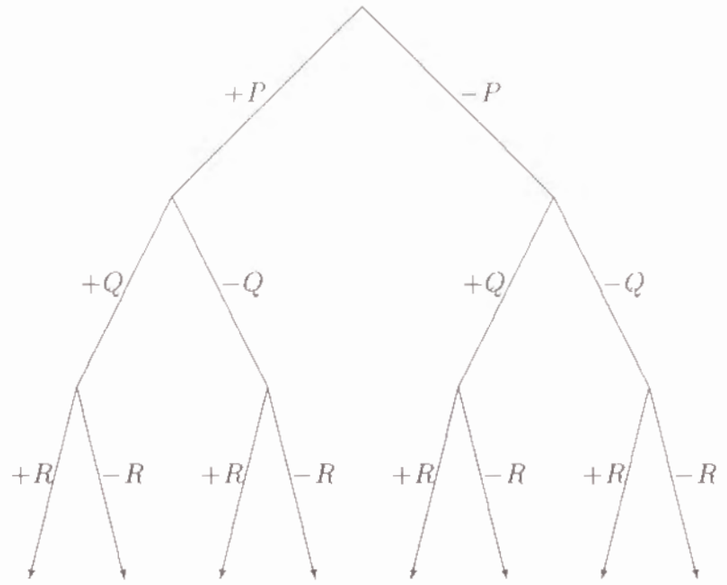


Figure 4.1: An example of complete semantic tree.

edges in the path, vice versa. For a node N in a complete semantic tree, we denotes the set of literals which label the edges in the path from the root to N as $I(N)$. Apparently, $I(N)$ is a partial H-interpretation.

Concerning to a complete semantic tree and partial H-interpretations, we give some definitions.

Definition 4.5 (failure uode, inference node) Let B be a complete semantic tree and let $\mathcal{S} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ be a set of clauses. A node N in B is called a failure node for $\mathcal{S}$ if and only if there exists a set of clauses $\mathcal{S}' = \{\mathcal{C}'_1, \cdots, \mathcal{C}'_n\}$ in which each $\mathcal{C}'_i$ is a ground instance of $\mathcal{C}_i$ and the $\mathcal{S}'$ is falsified by $I(N)$.

An inference node for $\mathcal{S}$ is a node which is not a failure node but all of whose descendant nodes are failure nodes. □

For the sake of convenience, for a set of clauses $\mathcal{S} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$, we denote a set of clauses $\mathcal{S}' = \{\mathcal{C}'_1, \cdots, \mathcal{C}'_n\}$ in which each $\mathcal{C}'_i$ is a ground instance of $\mathcal{C}_i$ as merely a ground instance of $\mathcal{S}$. And when there exists a ground instance of a clause $\mathcal{C}$ which is falsified by an H-interpretation I, we write merely $\mathcal{C}$ is falsified by I. Similarly, when there exists a ground instance of a set of clauses $\mathcal{S}$ which is falsified by an H-interpretation I, we write merely $\mathcal{S}$ is falsified by I.

Definition 4.6 (closed complete semantic tree) For a complete semantic tree B and a set of clauses $\mathcal{S}$ on a vocabulary, B is called closed if any downward path from the root is closed by failure node. □

With these definitions, we are able to prove the main theorem of this chapter. The outline of the proof is similar to that in first-order predicate logic described in Chang, et al.[Chang 73]. Before give the proof of the

theorem, we prove higher-order clausal logic versions of *Herbrand theorem* and *lifting lemma*.

Theorem 4.2 (Herbrand theorem on H-interpretation) Let $\mathcal{V}$ be a vocabulary which is restricted in the symbol types as described in theorem 4.1 and let $\mathcal{S}$ be a set of clauses on $\mathcal{V}$. Then, on the vocabulary $\mathcal{V}$, any complete semantic tree is closed for $\mathcal{S}$ if and only if $\mathcal{S}$ is falsified by any H-interpretation.

Proof. Assume that $\mathcal{S}$ is falsified by any H-interpretation. Let B be an arbitrary complete semantic tree and let P be an arbitrary downward path from the root of B. By the assumption, the H-interpretation which corresponds to the path P falsifies $\mathcal{S}$, and so there exists at least one ground instance of $\mathcal{S}$ which is falsified by the H-interpretation. Hence there exist a node N on the path P such that $I(N)$ falsifies the ground instance of $\mathcal{S}$ because the literals in the ground instance are finite. This holds for any B and P and so any complete semantic tree is closed for $\mathcal{S}$.

Conversely assume that any complete semantic tree is closed for $\mathcal{S}$. Let B be a complete semantic tree and let I is an arbitrary H-interpretation. Then there is a path P which corresponds to I in B. By the assumption, P includes a failure node. Hence $\mathcal{S}$ is falsified by any H-interpretation I. □

Lemma 4.1 (lifting lemma) Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be clauses and let σ be a substitution. If $\mathcal{C}'$ is a resolvent of $\mathcal{C}_1\sigma$ and $\mathcal{C}_2\sigma$, then there exist a substitution τ and a resolvent $\mathcal{C}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}\tau = \mathcal{C}'$.

Proof. Let:

$$\mathcal{C}_1\sigma = \mathcal{C}_{11}\sigma \cup \mathcal{C}_{12}\sigma \quad (\mathcal{C}_{11} \cup \mathcal{C}_{12} = \mathcal{C}_1)$$
$$\mathcal{C}_2\sigma = \mathcal{C}_{21}\sigma \cup \mathcal{C}_{22}\sigma \quad (\mathcal{C}_{21} \cup \mathcal{C}_{22} = \mathcal{C}_2)$$
$$\mathcal{C}' = (\mathcal{C}_{11}\sigma \cup \mathcal{C}_{22}\sigma)\,\pi \quad (\pi \text{ is an mgu of } \mathcal{C}_{12}\sigma \cup \mathcal{C}_{21}\sigma).$$

Then there exists an mgu θ of $\mathcal{C}_{12} \cup \mathcal{C}_{21}$ such that $\sigma \circ \pi = \theta \circ \tau$ for a substitution τ because $\sigma \circ \pi$ is a unifier of $\mathcal{C}_{12} \cup \mathcal{C}_{21}$. Let $\mathcal{C}$ be the resolvent of $\mathcal{C}_1$ and $\mathcal{C}_2$ with θ, that is:

$$\mathcal{C} = (\mathcal{C}_{11} \cup \mathcal{C}_{22})\,\theta$$

Hence:

$$\begin{aligned}\mathcal{C}\tau &= (\mathcal{C}_{11} \cup \mathcal{C}_{22})\,\theta \circ \tau \\ &= (\mathcal{C}_{11} \cup \mathcal{C}_{22})\,\sigma \circ \pi \\ &= \mathcal{C}'\end{aligned}$$

□

Now, we give a proof about completeness.

Theorem 4.3 (completeness for H-models) On a vocabulary which is restricted in the symbol types as described in theorem 4.1, a set of clauses $\mathcal{S}$ is refutable if and only if $\mathcal{S}$ is unsatisfiable for the model class of H-models.

Proof. Assume that a set of clauses $\mathcal{S}$ is unsatisfiable for the model class of H-models. Then, from corollary 4.1 and theorem 4.2, any complete semantic tree is closed for $\mathcal{S}$. Let B be an arbitrary complete semantic tree. Because B is closed, there exists at least one inference node in B if $\mathcal{S}$ does not contain an empty clause. So let N be an arbitrary inference node in B and let N_1, N_2 be the (direct) descendant nodes of N. Obviously, by the definition of complete binary tree, we can admit that $I(N_1) = I(N) \cup \{+\alpha\}$ and $I(N_2) = I(N) \cup \{-\alpha\}$ for an atomic formula α. Because N is not failure node but inference node, each of $\{+\alpha\}$ and $\{-\alpha\}$ falsifies $\mathcal{S}$. Therefore there exists a ground instance $\mathcal{C}_1'$ of a clause $\mathcal{C}_1$ in $\mathcal{S}$ which includes $-\alpha$, and there exists a ground instance $\mathcal{C}_2'$ of a clause $\mathcal{C}_2$ in $\mathcal{S}$ which includes $+\alpha$. In this

situation, $\mathcal{C}_1' \setminus \{-\alpha\}$ and $\mathcal{C}_2' \setminus \{+\alpha\}$ are falsified by $I(N)$ because literals in a clauses are disjunctive. Therefore a clause $\mathcal{C}' = (\mathcal{C}_1' \setminus \{-\alpha\}) \cup (\mathcal{C}_2' \setminus \{+\alpha\})$, which is a resolvent of $\mathcal{C}_1'$ and $\mathcal{C}_2'$, is also falsified by $I(N)$. From lemma 4.1, There exists a clause $\mathcal{C}$ which is a resolvent of $\mathcal{C}_1$ and $\mathcal{C}_2$ and of which the clause $\mathcal{C}'$ is an instance. Hence $I(N)$ is a failure node for the set of clauses $\mathcal{S} \cup \mathcal{C}$ which is derived from $\mathcal{S}$ with resolution principle.

By repeating this procedure, all nodes in B become failure nodes because B is finitely closed. When the root node of B becomes a failure node, derived set of clauses must include empty clause. Hence $\mathcal{S}$ is refutable with resolution principle.

Conversely, assume that there exists a H-model which satisfies the set of clauses $\mathcal{S}$. Then any clause in $\mathcal{S}$ is satisfied by the H-model. And so any resolvent of clauses in $\mathcal{S}$ is also satisfied by the H-model. Hence $\mathcal{S}$ never derive empty clause, that is, $\mathcal{S}$ is not refutable.

From the above the theorem has proved. □

4.4 Remarks and Discussions

In this chapter, we have defined an extension of Herbrand model on higher-order clausal logic as a model class with type restriction, and we have shown that the proof system based on resolution principle is complete for the model class. The completeness is significant because it is strongly related to correctness in logic programming system as mentioned in chapter 3.

Although this logic system is syntactically restricted, any term of first-order predicate logic is allowed. Moreover, a term of any type which is constructed solely from 1 is also allowed.

These features imply that we are able to extend (first-order) logic pro-

gramming system with the logic system. For example, a kind of functional calculation can be introduced in logic programming system.

Example 4.1 Let $\tilde{n}$ denote $\lambda fx.(\overbrace{f(f\cdots(f}^{n}x)\cdots))$ for any $n \geq 0$ where $f \in V_{(11)}$ and $x \in V_1$. And let μ, ν denote $\lambda uv.(\lambda fx.u(vf)x)$ and $\lambda u.(\lambda fx.f(ufx))$ respectively where $u, v \in V_{\langle(11)\langle(11)\rangle\rangle}$, $f \in V_{(11)}$ and $x \in V_1$. Then $\mu\tilde{n}\tilde{m} \overset{\alpha,\beta}{=} \widetilde{nm}$ and $\nu\tilde{n} \overset{\alpha,\beta}{=} \widetilde{n+1}$. This means μ and ν have a function as multiplication and succession to natural numbers embedded in lambda expressions.

Using this natural number embedding, we can define a predicate which denotes factorial as a set of clauses in higher-order clausal logic consists of the following two clauses[6]:

1: $\{+F\tilde{0}\tilde{1}\}$

2: $\{+F(\nu p)(\mu(\nu p)q), -Fpq\}$

where $F \in T_{\langle 1\langle 10\rangle\rangle}$ and $p, q \in T_1$.

Let the following clause is a goal clause:

3: $\{-F\tilde{1}w\}$

where $w \in T_1$.

By expanding μ's, ν's and embedded numbers (and by normalizing), these clauses are equivalent to the following clauses:

1′: $\{+F(\lambda fx.x)(\lambda fx.fx)\}$

2′: $\{+F(\lambda fx.f(pfx))(\lambda fx.qf(p(qf)x)), -Fpq\}$

3′: $\{-F(\lambda fx.fx)w\}$

Here, we give a refutation from these clauses:

[6]Each atomic formula in this expression does not look like normal form. However it is no matter because each formula is a term class.

4: $\{-F(\lambda fx.x)q\}$, (from 3′ and 2′ with $\theta_1 = \{p \leftarrow \lambda fx.x, w \leftarrow \lambda fx.qfx\}$

5: $\{\}$, (from 4 and 2′ with $\theta_2 = \{q \leftarrow \lambda fx.fx\}$

Composing the substitutions in the refutation, we get a solution $w = \tilde{1}$.

□

On the other hand, it is unknown that the type restriction is "critical". It is possible that there are more loose syntactic constraints. We hope, however, the type restriction described in this chapter be sufficient for some applications because of above features. Anyway, syntactic constraint is not preferable even if it is loose. In the next chapter, we consider a model class and completeness for the constraint free higher-order clausal logic.

Chapter 5

Higher-Order Clausal Logic without Type Restriction

5.1 Introduction to Chapter 5

As mentioned in chapter 3, straightforward extension of (first-order) Herbrand model to Higher-order clausal logic does not succeed. In the last chapter, we have made it by restricting types in the vocabulary. However, any kind of restriction is not so desirable. So it is significant if we define a model class such that, without any syntactic restriction, each element in its domain is expressible in the vocabulary such as Herbrand model. To realize this notion, we must consider the problem described in example 3.1 again. In the example, (RP) and (RQ) are distinct expressions and corresponding elements in H-universe are also distinct. From functional view, however, if P and Q have a same value 1 in a g-model, (RP) and (RQ) must have a same value in the g-model. So the domains of such a g-model must not be isomorphic with H-universes.

Hence, in this chapter, we define a class of g-models called denotational models, in which each domain is isomorphic with a quotient set of the H-universe of same type based on a equivalence relation such as (RP) and (RQ) in example 3.1 are regarded equivalent. In such a model class, each domain is not isomorphic with corresponding H-universe any more but homomorphic with it. Moreover, any element in the domain still has corresponding expressions in the H-universe. This feature is desirable for considering correctness of logic programming system as discussed in chapter 3. Furthermore, if the vocabulary is restricted to first-order, this model class becomes the class of Herbrand models.

On the other hand, this model class arises another question — is the proof system based on resolution principle complete on such a model class? The answer of this question is "no". Afterwards, we show an example, a set of clauses which is unsatisfiable on the model class but which is not refutable. This fact is concerned with the equivalence classes in H-universes introduced in the model class. The proof system only based on resolution principle does not have any way to infer the equality of values of terms in one equivalence class.

So, in this chapter, we also introduce some axioms about equality to our proof system, by which the proof system becomes complete on the model class.

5.2 Denotational Model for Higher-Order Clausal Logic

A model class defined in this chapter will be called a class of denotational models. As the preparations to define a denotational model, we give some definitions. We also use notions of H-universe, H-base and so on defined in chapter 4.

First of all, we define an equivalence relation among term classes.

Definition 5.1 (equivalence relation $\sim$) An equivalence relation among term classes is defined as followings recursively depending on an H-interpretation or a partial H-interpretation I:

1. For any positive literals $+\alpha$ and $+\beta$ in I, $\alpha \sim \beta$.
2. For any negative literals $-\alpha$ and $-\beta$ in I, $\alpha \sim \beta$.
3. For any term class α, $\alpha \sim \alpha$.
4. For any term classes α, β, γ and δ, $(\alpha\gamma) \sim (\beta\delta)$ if $\alpha \sim \beta$ and $\gamma \sim \delta$.
5. For any term classes α and β and for any variable x, $(\lambda x.\alpha) \sim (\lambda x.\beta)$ if $\alpha \sim \beta$.
6. For any term classes α, β and γ, $\alpha \sim \gamma$ if $\alpha \sim \beta$ and $\beta \sim \gamma$.
7. Only in pairs derived from the above conditions recursively, the relation $\sim$ holds.

□

This relation is regarded as the equivalence which is consequent on the equivalence in H-base depending on a partial or total H-interpretation. Notice that the number of elements in $H_0/\sim$ for a total H-interpretation is at most 2, that is obvious from 1 and 2 in the above definition.

From another viewpoint, this relation means a constraint among term classes which should have a same value. This constraint is applied also in H-base. So, if underlying H-interpretation is not adequate, the constraint forces inconsistent situation, in which α and β should have a same value for some $+\alpha$ and $-\beta$ in the H-interpretation. To distinguish such an inadequate H-interpretation, we introduce a notion *regularity*.

Definition 5.2 (regularity of H-interpretation) For an H-interpretation (or a partial H-interpretation) I, if there exist two literals $+\alpha$ and $-\beta$ in I such that $\alpha \sim \beta$ then the (partial) H-interpretation is called *irregular*. Otherwise, the (partial) H-interpretation is called *regular*. □

Considering 1, 2 and 6 in definition 5.1, the number of elements in $H_0/\sim$ is 1 if underlying H-interpretation is a total and irregular one. On the other hand, it is also obvious that the number of elements in $H_0/\sim$ is just 2 if underlying H-interpretation is total and regular one.

Under the above preparation, we are able to define a denotational model, a g-model whose g-frame is isomorphic to $[H_t/\sim]_{t\in T}$.

Definition 5.3 (denotational model) For a regular total H-interpretation I, a g-model $M = [D_t, m_t]_{t\in T}$ is called denotational model if there is a family of isomorphism $[\Psi_t^I]_{t\in T}$ such that:

$$\Psi_t^I : H_t/\sim \longrightarrow D_t, \quad (t \in T)$$

and which satisfies following conditions:

1. For any positive literal $+\alpha$ in I, $\Psi_0^I[[\alpha]] = 1$. ($[\alpha]$ means a equivalence class which includes α)
2. For any negative literal $-\beta$ in I, $\Psi_0^I[[\beta]] = 0$.
3. For any α in $H_{(st)}$ and β in H_s, $\Psi_t^I[[(\alpha\beta)]] = (\Psi_{(st)}^I[[\alpha]])[\Psi_s^I[[\beta]]]$.
4. For any constant A in H_t, $m_t[A] = \Psi_t^I[[A]]$.

□

Apparently, exact one family of isomorphism $[\Psi_t^I]_{t\in T}$ exists for a regular total H-interpretation I. In other words, the isomorphism connects a regular total H-interpretation, a set of closed literals which should be true, to a g-model, a family of pairs each of which consists of a domain and a meaning function, in one-to-one way. Because of the isomorphism, the following properties hold immediately.

Property 5.1 Let M be a denotational model corresponding to a regular total H-interpretation I, and let $[\Psi_t^I]_{t\in T}$ be the isomorphism described in definition 5.3. Then, $V_M[\alpha] = \Psi_t^I[[\alpha]]$ for any term class α in H_t. And so, for any literal L in a regular total H-interpretation I and the denotational model M corresponding to I, $V_M[L] = 1$. □

Under the definition of a denotational model, the set of literals in example 3.1 in chapter 3 becomes a total but irregular H-interpretation. And so, no denotational model corresponds to the H-interpretation. An irregular H-interpretation causes contradictions in interpretation of expressions as shown in example 3.1. When we consider only denotational models, however, we can avoid the contradictions because no denotational model corresponds to such a irregular H-interpretation.

For a fixed vocabulary, entire denotational models each of which corresponds to a regular total H-Interpretation form a model class. In the rest of this chapter, we consider completeness of our proof system on the model class.

5.3 Extension of Proof System with Equality Axioms

As mentioned above, the proof system with resolution principle which has described in chapter 2 is not complete on the class of denotational models. Example 5.1 is an example of such a set of clauses.

Example 5.1

$$\mathcal{S} = \{\{+P\}, \{+Q\}, \{+(RP)\}, \{-(RQ)\}\}$$

□

Obviously, any H-interpretation which satisfies the set of clauses in this example is irregular. And so, no denotational model satisfies it[1]. However, it is impossible to refute this set of clauses only with resolution principle. The reason of it is that the proof system has no way to infer that the values of P and Q are equal, and so the values of (RP) and (RQ) are also equal, although it can prove that both P and Q is true. In other words, the proof system cannot exclude irregular H-interpretation which include $\mathcal{S}$, and so which satisfies $\mathcal{S}$.

[1]Obviously no g-model satisfies the set of clauses, either. Therefore, this example is a counter example for g-completeness of resolution principle asserted in Pietrzykowski[Pietrzykowski 72].

Hence, if we want to let the proof system be complete, we must extend the proof system with more inference rules or axioms which make it possible to infer the value equality. In this section we define an extended proof system which has extended vocabulary and some axioms, and which is able to infer the value equality.

Extended vocabulary is obtained by adding two families of constant symbols. One of them is a family of binary predicate which means equality, and the other is a family of auxiliary functions.

Definition 5.4 (extended vocabulary) Let $\mathcal{V}$ be a vocabulary. Then, the vocabulary which is obtained by adding each of the following constants to $\mathcal{V}$ is called extended vocabulary and denoted as $\mathcal{V}^+$.

1. $\equiv_t$: a constant of type $\langle t\langle t0\rangle\rangle$ (for any $t \in T$).
2. $\Gamma_{\langle st\rangle}$: a constant of type $\langle\langle st\rangle\langle\langle st\rangle s\rangle\rangle$ (for any $s, t \in T$).

□

From the viewpoint of model theory, $\equiv_t$ means the equality in D_t which corresponds to $\sim$ in H_t, and $\Gamma_{\langle st\rangle}$ is a Skolem function whose value is a function which takes two arguments f and g of type $\langle st\rangle$ and returns a value d of type s such that $f[d] \neq g[d]$ if $f \neq g$. For convenience, $\equiv_t$ is used in infix notation as usual, and subscriptions of these symbols may be omitted.

To implement the meanings of the new constants in the proof system, we introduce the following axiom schemata. The Skolem functions $\Gamma_{\langle st\rangle}$ are needed in the axiom schema **E5**.

Definition 5.5 (equality axioms) The following clauses are the equality axiom schemata. Any clause which has the same figure with one of the

following schemata is an axiom. Some of the following schemata derive a lot of axioms according to a lot of distinct $\equiv$'s and Γ's.

E1: $\{+(x \equiv y), -x, -y\}$

E2: $\{+(x \equiv y), +x, +y\}$

E3: $\{+(x \equiv x)\}$

E4: $\{+((xz) \equiv (yw)), -(x \equiv y), -(z \equiv w)\}$

E5: $\{+(x \equiv y), -((x(\Gamma xy)) \equiv (y(\Gamma xy)))\}$

E6: $\{+(x \equiv z), -(x \equiv y), -(y \equiv z)\}$

E7: $\{-(x \equiv y), -x, +y\}$

E1, **E2** and **E7** are only used for $\equiv_0$, although **E3**, **E4**, **E5** and **E6** are used for any $\equiv_t$. For the sake of convenience, the set of all axioms derived from these schemata may be denoted as $\mathcal{E}$. □

The axiom schemata from **E1** to **E6** correspond to the definition of $\sim$ in section 5.2, and **E7** means regularity.

By using this axiom schemata, the proof system with resolution principle are extended.

Definition 5.6 (deduction, refutation in extended proof system) Let $\mathcal{S}$ be a set of clauses on a vocabulary $\mathcal{V}$. A sequence of clauses $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n$ is called deduction of $\mathcal{C}$ from $\mathcal{S}$ in extended proof system if each $\mathcal{C}_i$ satisfies the following conditions:

1. Each $\mathcal{C}$ is a clause on the vocabulary $\mathcal{V}^+$.

2. Each $\mathcal{C}$ is a clause in $\mathcal{S} \cup \mathcal{E}$ or a resolvent of $\mathcal{C}_j$ and $\mathcal{C}_k$ such that $j, k < i$.

3. $\mathcal{C}_n = \mathcal{C}$.

A deduction of empty clause from $\mathcal{S}$ in extended proof system is called refutation of $\mathcal{S}$ in extended proof system. □

5.4 Completeness for Denotational Model

By extended proof system defined in the last chapter, we can refute any set of clauses which is not satisfied by any denotational model. We prove it in this section. The way of the proof is similar to that in chapter 4 except the treatments of irregular H-interpretation and equality axioms.

First of all we prove a lemma which shows a connection between $\sim$ and $\equiv$.

Lemma 5.1 Let α and β be two term classes of a same type on a vocabulary $\mathcal{V}$. Then if $\alpha \sim \beta$ for any partial or total H-interpretation I on $\mathcal{V}$, there exist an substitution θ and clauses $\mathcal{C}$ and $\mathcal{C}'$ which satisfy following conditions:

1. $\mathcal{C}$ can be deduced from $\mathcal{E}$ with resolution principle.

2. $\mathcal{C}\theta = \mathcal{C}' \cup \{+(\alpha \equiv \beta)\}$.

3. $\mathcal{C}'$ is falsified by I.

4. $\mathcal{C}'$ has no free variable.

Proof. We prove it by mathematical induction according to the definition of $\sim$.

1. In the case of $+\alpha, +\beta \in I$.

   Let $\mathcal{C}$ be $\{-x, -y, +(x \equiv y)\}$ derived immediately from E1 and θ be $\{x \leftarrow \alpha, y \leftarrow \beta\}$. Then $\mathcal{C}\theta = \{-\alpha, -\beta\} \cup \{+(\alpha \equiv \beta)\}$. So $\mathcal{C}'$ is $\{-\alpha, -\beta\}$, which is falsified by I because $+\alpha, +\beta \in I$.

2. In the case of $-\alpha, -\beta \in I$.

   It can be proved in a similar way to 1 with E2.

3. In the case of $\beta = \alpha$ i.e. $\alpha \sim \alpha$.

   Let $\mathcal{C}$ be $\{+(x \equiv x)\}$ derived from E3 and θ be $\{x \leftarrow \alpha\}$. Then $\mathcal{C}\theta = \emptyset \cup \{+(\alpha \equiv \alpha)\}$. So $\mathcal{C}'$ is $\emptyset$, which is falsified by I.

4. In the case of $\alpha = (\alpha_1\alpha_2)$, $\beta = (\beta_1\beta_2)$, $\alpha_1 \sim \beta_1$ and $\alpha_2 \sim \beta_2$.

   From the assumption of induction:

$$\mathcal{C}_1\theta_1 = \mathcal{C}_1' \cup \{+(\alpha_1 \equiv \beta_1)\}$$
$$\mathcal{C}_2\theta_2 = \mathcal{C}_2' \cup \{+(\alpha_2 \equiv \beta_2)\}$$

   where $\mathcal{C}_1$ and $\mathcal{C}_2$ can be deduced from $\mathcal{E}$ and $\mathcal{C}_1'$ and $\mathcal{C}_2'$ are ground clauses falsified by I. Then $\mathcal{C}' = \mathcal{C}_1' \cup \mathcal{C}_2' \cup \{+((\alpha_1\alpha_2) \equiv (\beta_1\beta_2))\}$ can be deduced from $\mathcal{C}_1\theta_1$, $\mathcal{C}_2\theta_2$ and E4 with substitution $\{x \leftarrow \alpha_1, y \leftarrow \beta_1, z \leftarrow \alpha_2, w \leftarrow \beta_2, \}$. Obviously, $\mathcal{C}_1' \cup \mathcal{C}_2'$ are falsified by I and, from lemma 4.1, there exists a clause $\mathcal{C}$ deduced from $\mathcal{C}_1$, $\mathcal{C}_2$ and E4 and a substitution θ such that $\mathcal{C}\theta = \mathcal{C}'$.

5. In the case of $\alpha = (\lambda w.\alpha_1)$, $\beta = (\lambda w.\beta_1)$ and $\alpha_1 \sim \beta_1$.

   From the assumption of induction:

$$\mathcal{C}_1\theta_1 = \mathcal{C}_1' \cup \{+(\alpha_1 \equiv \beta_1)\}$$

   where $\mathcal{C}_1$ can be deduced from $\mathcal{E}$ and $\mathcal{C}_1'$ are ground clauses falsified by I. Then $\mathcal{C}' = \mathcal{C}_1' \cup \{+((\lambda w.\alpha_1) \equiv (\lambda w.\alpha_1))\}$ can be deduced from $\mathcal{C}_1\theta_1$ and E5 with substitution $\{x \leftarrow (\lambda w.\alpha_1), y \leftarrow (\lambda w.\beta_1), w \leftarrow (\Gamma(\lambda w.\alpha_1)(\lambda w.\beta_1))\}$. From lemma 4.1, there exists a clause $\mathcal{C}$ deduced from $\mathcal{C}_1$ and E5 and a substitution θ such that $\mathcal{C}\theta = \mathcal{C}'$.

6. In the case of $\alpha \sim \gamma$ and $\gamma \sim \beta$ for some γ.

   From the assumption of induction:

$$\mathcal{C}_1\theta_1 = \mathcal{C}_1' \cup \{+(\alpha \equiv \gamma)\}$$
$$\mathcal{C}_2\theta_2 = \mathcal{C}_2' \cup \{+(\gamma \equiv \beta)\}$$

   where $\mathcal{C}_1$ and $\mathcal{C}_2$ can be deduced from $\mathcal{E}$ and $\mathcal{C}_1'$ and $\mathcal{C}_2'$ are ground clauses falsified by I. Then $\mathcal{C}' = \mathcal{C}_1' \cup \mathcal{C}_2' \cup \{+(\alpha \equiv \beta)\}$ can be deduced from $\mathcal{C}_1\theta_1$, $\mathcal{C}_2\theta_2$ and E6 with substitution $\{x \leftarrow \alpha, y \leftarrow \gamma, z \leftarrow \beta, \}$. Obviously, $\mathcal{C}_1' \cup \mathcal{C}_2'$ are falsified by I and, from lemma 4.1, there exists a clause $\mathcal{C}$ deduced from $\mathcal{C}_1$, $\mathcal{C}_2$ and E6 and a substitution θ such that $\mathcal{C}\theta = \mathcal{C}'$.

From 1 to 6, The lemma holds inductively. □

From this lemma, the following corollary about irregular H-interpretation holds immediately.

Corollary 5.1 Let I be an arbitrary irregular partial or total H-interpretation. There exists a clause which is falsified by I and which can be deduced from $\mathcal{E}$.

Proof. Because I is irregular, there exist two atomic formulae α, β such that $+\alpha, -\beta \in I$ and $\alpha \sim \beta$. Then, from lemma 5.1, there exists a clause $\mathcal{C}_0$ which can be deduced from $\mathcal{E}$ and which holds:

$$\mathcal{C}_0\theta_0 = \mathcal{C}_0' \cup \{+(\alpha \equiv \beta)\}$$

with a substitution θ_0, where $\mathcal{C}'_0$ is falsified by I. And so the following clause is deduced from E7 and $\mathcal{C}_0\theta_0$ with substitution $\{x \leftarrow \alpha, y \leftarrow \beta\}$:

$$\mathcal{C}'_0 \cup \{-\alpha, +\beta\}$$

This clause is falsified by I because $+\alpha, -\beta \in I$. On the other hand, from lemma 4.1, there exists a clause $\mathcal{C}$ which can be deduced from E7 and $\mathcal{C}_0$ and which holds:

$$\mathcal{C}\theta = \mathcal{C}'_0 \cup \{-\alpha, +\beta\}$$

with a substitution θ. This $\mathcal{C}$ holds the conditions of this corollary. □

Under the above preparation, we are able to prove denotational model version of Herbrand theorem. As in chapter 4, we use a complete semantic tree. With the type restriction in chapter 4, any downward path from the root to a leaf corresponds to a (regular) H-interpretation. However, without any type restriction, some of such paths may correspond to irregular H-interpretations. Hence we extend the definitions of failure node and inference node.

Definition 5.7 (failure node, inference node (extended version)) For a complete semantic tree B and a set of clauses $\mathcal{S}$ on a vocabulary, a node N in B is a failure node for $\mathcal{S}$ if and only if $I(N)$ is irregular or $I(N)$ falsifies $\mathcal{S}$.

An inference node for $\mathcal{S}$ is a node which is not a failure node but all of whose descendant nodes are failure nodes. □

Now, we prove completeness of extended proof system for the model class of denotational model after the proof of denotational model version of Herbrand theorem.

Theorem 5.1 (Herbrand theorem on denotational model) Let $\mathcal{V}$ be a vocabulary and let $\mathcal{S}$ be a set of clauses on $\mathcal{V}$. Then, on the vocabulary $\mathcal{V}$, any complete semantic tree is closed for $\mathcal{S}$ if and only if $\mathcal{S}$ is falsified by any denotational model.

Proof. Assume that $\mathcal{S}$ is falsified by any denotational model. Let B be an arbitrary complete semantic tree, let P be an arbitrary downward path from the root of B, let I be the H-interpretation which corresponds to P and let M be the denotational model which corresponds to I.

If I is irregular, from the definition of irregularity, there are two literals $+\alpha, -\beta \in I$ and $\alpha \sim \beta$. This $\sim$ relation has been derived with a finite application of rule 4 - 6 from rule 1 - 3 in definition 5.1. Therefore there is a finite set of literals which is a foundation of $\alpha \sim \beta$. Hence there is a node N of P such that $I(N)$ includes the literals and also includes $+\alpha$ and $-\beta$. Apparently N is a failure node.

If I is regular, by the assumption, $V_M[\mathcal{S}] = 0$. Therefore there is at least one clause $\{L_1, \cdots, L_m\}$ in $\mathcal{S}$ and $V_{M\mathbf{a}}[L_1] = \cdots = V_{M\mathbf{a}}[L_m] = 0$ for a family of assignments $\mathbf{a} = [\mathrm{a}_t]_{t \in T}$. Because H-universes and domains of M isomorphic, there is a ground clause $\{L_1, \cdots, L_m\}\theta$ which is falsified by I where $\theta = \{\alpha^t \leftarrow x^t \mid \exists \alpha^t \in \Psi_t^{I^{-1}}[\mathrm{a}_t[x^t]]$ for any free variable x^t in $L_1, \cdots, L_m\}$ This ground clause is a finite set of at most n literals. Therefore there is a node N of P such that $I(N)$ falsifies the ground clause, and so it falsifies $\mathcal{S}$. Hence N is a failure node.

Since there is a failure node on any P, B is closed for $\mathcal{S}$

Conversely assume that any complete semantic tree is closed for $\mathcal{S}$. Let B be a complete semantic tree, let M is an arbitrary denotational model and let I is the H-interpretation which corresponds to M. Then there is

a path P which corresponds to I in B. Since B is closed for $\mathcal{S}$, there is a node N on P and $I(N)$ falsifies a ground instance of $\mathcal{S}$. Obviously I also falsifies $\mathcal{S}$ □

Theorem 5.2 (completeness for denotational models) a set of clauses $\mathcal{S}$ on a vocabulary $\mathcal{V}$ is refutable with extended proof system if and only if $\mathcal{S}$ is unsatisfiable for the model class of denotational models.

Proof. Assume that a set of clauses $\mathcal{S}$ is unsatisfiable for the model class of denotational model. Then, from theorem 5.1, any complete semantic tree is closed for $\mathcal{S}$. Let B be an arbitrary complete semantic tree. Because B is closed, there exists at least one inference node in B if $\mathcal{S}$ does not contain an empty clause. So let N be an arbitrary inference node in B and let N_1, N_2 be the (direct) descendant nodes of N. Obviously, by the definition of complete binary tree, we can admit that $I(N_1) = I(N) \cup \{+\alpha\}$ and $I(N_2) = I(N) \cup \{-\alpha\}$ for an atomic formula α.

If $I(N_1)$ is irregular, from corollary 5.1, there exists a clause which can be deduced from $\mathcal{E}$ and which is falsified by $I(N_1)$. Let C_1 be the clause and let C_1' be a ground instance of C_1 which is falsified by $I(N_1)$. Then $-\alpha \in C_1'$ because, if we assume $-\alpha \notin C_1'$, $I(N)$ also becomes irregular, which is contradict to assumption.

On the other hand, if $I(N_1)$ is regular, there exists a clause in $\mathcal{S}$ which is falsified by $I(N_1)$. Let C_1 be the clause and let C_1' be a ground instance of C_1 which is falsified by $I(N_1)$. In this case, $-\alpha \in C_1'$, too because N is not failure node but inference node.

For the node N_2, there exists a ground instance C_2' of a clause C_2 in $\mathcal{S}$ and $+\alpha \in C_1'$ as for N_1.

In this situation, $C_1' \setminus \{-\alpha\}$ and $C_2' \setminus \{+\alpha\}$ are falsified by $I(N)$ because

literals in a clauses are disjunctive. Therefore a clause $C' = (C_1' \setminus \{-\alpha\}) \cup (C_2' \setminus \{+\alpha\})$, which is a resolvent of C_1' and C_2', is also falsified by $I(N)$. From lemma 4.1, There exists a clause C which is a resolvent of C_1 and C_2 and of which the clause C' is an instance. Obviously C can be deduced from $\mathcal{S}$ by extended proof system. and $I(N)$ becomes a failure node for the set of clauses $\mathcal{S} \cup C$ which is derived from $\mathcal{S}$ with resolution principle and equality axioms.

By repeating this procedure, all nodes in B become failure nodes because B is finitely closed. When the root node of B becomes a failure node, derived set of clauses must include empty clause. Hence $\mathcal{S}$ is refutable by extended proof system.

Conversely, assume that there exists a denotational model which satisfies the set of clauses $\mathcal{S}$. Let I be the H-interpretation which corresponds to the denotational model, which is on $\mathcal{V}$. Then we can construct a total H-interpretation I^+ on $\mathcal{V}^+$ which satisfies following conditions.

1. $I^+ \supset I$.
2. if $\alpha \sim \beta$ then $I^+ \ni +(\alpha \equiv \beta)$.
3. if $\alpha \not\sim \beta$ then $I^- \ni +(\alpha \equiv \beta)$.
4. if $\alpha \sim \beta$ and $I^+ \ni \pm\gamma\{\alpha\}$ then $I^+ \ni \pm\gamma\{\beta\}$, where $\gamma\{\alpha\}$ means an atomic formula containing α as its subterm and $\gamma\{\beta\}$ means the atomic formulae the subterm replaced with β.

Let M^+ be the denotational model which corresponds to I^+.

Then $V_{M^+}[\mathcal{S} \cup \mathcal{E}] = 1$, and so any clause deduced from $\mathcal{S} \cup \mathcal{E}$ is also satisfied by M^+ Hence $\mathcal{S}$ cannot deduce empty clause with extended proof system, that is, $\mathcal{S}$ is not refutable with extended proof system.

From the above the theorem has proved. □

The proofs of lemma 5.1, corollary 5.1 and theorem 5.2 also give a plan for refutation. Example 5.2 shows a refutation steps from the set of clauses of example 5.1

Example 5.2 Let $\mathcal{S}$ be the set of clauses $\{\{+P\},\{+Q\},\{+(RP)\},\{-(RQ)\}\}$. Figure 5.1 is an example of complete semantic tree for $\mathcal{S}$.

In this complete semantic tree, N_3 is an inference node and N_1 and N_2 are failure nodes. Because $I(N_2)$ is irregular, we get the following deduction.

1: $\{+((xz) \equiv (xw)), -(z \equiv w)\}$ (from E3, E4)
2: $\{+((xz) \equiv (xw)), -z, -w\}$ (from E1, 1)
3: $\{-(xz), +(xw), -z, -w\}$ (from 2, E7)

Then the clause numbered with 3 is falsified by $I(N_1)$. On the other hand, $I(N_1)$ falsifies $\{-(RQ)\}$. Hence we can continue the deduction.

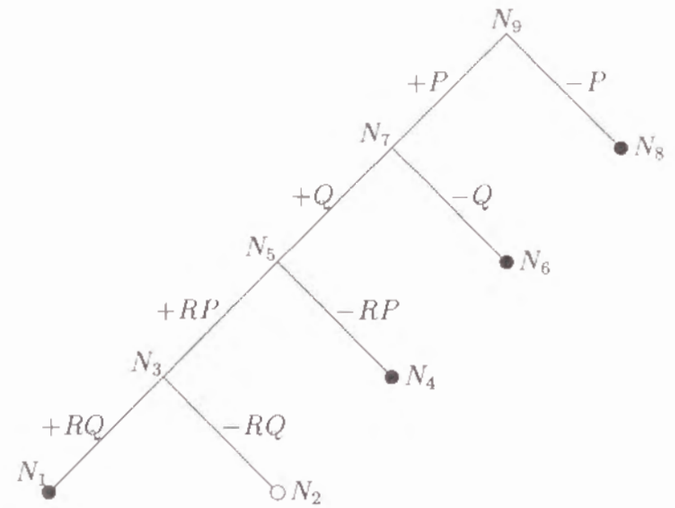
4: $\{-(RQ)\}$ (in $\mathcal{S}$)
5: $\{-(Rz), -z, -Q\}$ (from 3, 4)

This clause numbered with 5 is falsified by $I(N_3)$. So N_5 becomes inference node. Because $I(N_4)$ falsifies $\{+(RP)\}$, we can continue.

6: $\{+(RP)\}$ (in $\mathcal{S}$)
7: $\{-P, -Q\}$ (from 5, 6)

Then N_7 becomes inference node and $I(N_6)$ falsifies $\{+Q\}$.

8: $\{+Q\}$ (in $\mathcal{S}$)
9: $\{-P\}$ (from 7, 8)



● failure nodes for $\mathcal{S}$
○ failure node with irregular partial interpretation
(Subtrees lower than failure nodes are omitted.)

Figure 5.1: An example of closed complete semantic tree for example 5.2

Finally, N_9 becomes inference node and we can complete refutation.

10: $\{+P\}$ (in $\mathcal{S}$)
11: $\{\}$ (from 9, 10)

□

5.5 Remarks and Examples

In this chapter, we have defined the class of denotational models and shown that the proof system with resolution principle and equality axioms is com-

plete for the model class. A denotational model discussed in this chapter has some preferable features. First, it is a natural extension of a Herbrand model in first-order predicate logic. Secondly, it can be defined on any vocabulary dissimilarly to H-model described in chapter 4. Lastly, any element in whose domains has at least one corresponding expression in H-universes. The last feature is preferable when we consider higher-order clausal logic as a logic programming system because it seems to be appropriate to exclude an element from the domains which has no corresponding expression. and it is strongly related to correctness in logic programming system as mentioned in chapter 3.

Obviously, H-model and denotational model are strongly related each other. Indeed, it is apparent that H-model and denotational model are isomorphic under the type restriction introduced in chapter 4. Figure 5.2 shows the relations between standard model, generalized model, H-model (Herbrand model) and denotational model.

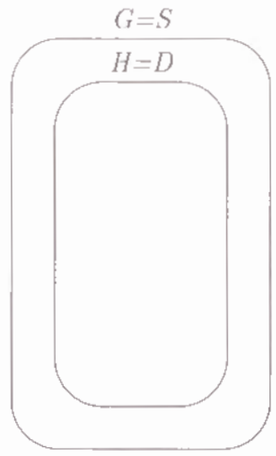
When we regard higher-order clausal logic as an extended logic programming system, it has great richness for expressing many things. The following examples present some of the applications.

Example 5.3 (function composition) In logic programming, unary functions are usually expressed as a binary predicate which represents the relation of the input and the output of the function, for example:

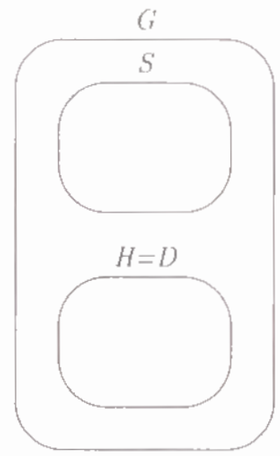
$$y = f(x) \quad \longrightarrow \quad +(Fxy)$$

Therefore, function composition is represented as the following:

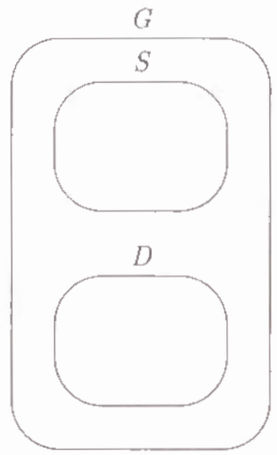
$$y = f \circ g(x) = f(g(x)) \quad \longrightarrow \quad +(Fxz) and + (Gzy)$$



(a) for first-order predicate logic



(b) for higher-order clausal logic with type restriction



S : model class of standard model
G : model class of generalized model
H : model class of H-model (Herbrand model)
D : model class of denotational model

(c) for higher-order clausal logic without type restriction

Figure 5.2: Relationships between model classes

In this representation, there is no expression corresponding to $f \circ g$ in logic programming.

In higher-order clausal logic, we can represent it in more smart way. A metapredicate C can be defined with following clause.

$$\{+(Cpqxy), -(qxz), -(pzy)\}$$

Then (CFG) means the composition of F and G. Indeed, with this clause, two goal clauses $\{-(CFGAy)\}$ and $\{-(GAz), -(Fzy)\}$ are equivalent. □

Example 5.4 (term rewriting system) The following set of clauses $\mathcal{S}$ can simulate a term rewriting system for a given set R of term rewriting rules.

$$\mathcal{S} = \{\ \{+(R\alpha\beta)\} \mid \alpha \rhd \beta \in R\}$$
$$\cup\{\ \{+(px), -(py), -(Rxy)\}, \{+(Exx)\}\ \}$$

let R be $\{(AZx) \rhd x, (A(Sx)y) \rhd (S(Axy))\}$, which represents addition of natural numbers[2]. Then:

$$\mathcal{S} = \{\ \{+(R(AZx)x)\}, \{+(R(A(Sx)y)(S(Axy)))\}\ \}\ \}$$
$$\cup\{\ \{+(px), -(py), -(Rxy)\}, \{+(Exx)\}\ \}$$

This set of clauses can simulate the given term rewriting system. For example, $\mathcal{S} \cup \{\ \{-(Ex(A(SZ)(SZ)))\}\ \}$ can be refuted by extended proof system and, by composing mgu's, we can get the solution $x = (S(SZ))$[3].

The simulated term rewriting system can be used in logic programs. For example, $\mathcal{S}$ added with the following clauses works as a program for

[2] A, S and Z represent addition, successor and zero respectively, and, in the rest of this example, M and F represents multiplication and factorial.

[3] This kind of solution is not always normal form. Normalization strategy is the other problem.

multiplication.

$$\{+(MZyZ)\}$$
$$\{+(M(Sx)yz), -(Mxyw), -(Ez(Ayw))\}$$

By adding further the following clauses, the set of clauses works a program for factorial.

$$\{+(R(FZ)(SZ))\}$$
$$\{+(R(Sx)y), -(M(Sx)(Fx)y)\}$$

□

Higher-order clausal logic can be applied to a metareasoning system for logic programming systems. as well as an extended logic programming system. The reasoning system can use mathematical induction unlike reasoning systems on first-order predicate logic.

Example 5.5 The following clauses represents mathematical induction[4].

1: $\{+(pw), +(p(Xp)), -(pZ)\}$
2: $\{+(pw), -(p(S(Xp))), -(pZ)\}$

This clauses are derived from the following logical expression.

$$(\forall p)(((pZ) \wedge (\forall x)((px) \supset (p(Sx)))) \supset (\forall w)(pw))$$

From the clauses with a logic program for addition, we can prove $+(AwZw)$ which means $w + 0 = w$.

3: $\{+(AZyy)\}$ $(0 + y = y)$
4: $\{+(A(Sx)y(Sz)), -(Axyz)\}$ (if $x + y = z$ then $(x + 1) + y = z + 1$)

[4] Z, S and X represents zero, successor and a Skolem function.

5: $\{+(AwZw), +(A(X(\lambda v.(AvZv)))Z(X(\lambda v.(AvZv))))\}$
(from 3 and 1)

6: $\{+(AwZw), -(A(S(X(\lambda v.(AvZv))))Z(S(X(\lambda v.(AvZv)))))\}$
(from 3 and 2)

7: $\{+(AwZw), +(A(S(X(\lambda v.(AvZv))))Z(S(X(\lambda v.(AvZv)))))\}$
(from 5 and 4)

8: $\{+(AwZw)\}$ (from 7 and 6)

□

With equality axioms, we extend our proof system in this chapter. However, it is not clear that this extension is redundant or not, or there are any other extension, for example, not axioms but inference rules. These problems are left for future works.

Axiomatic method we adapted in this chapter has further possibilities. Another application of the method is to introduce other logical symbols such as conjunction, disjunction or implication. The problems how the logic and proof system behave are also left for future works.

Chapter 6

Parallel Implementation of the Proof Systems

6.1 Introduction to Chapter 6

In section 2.3, we described a proof system for higher-order clausal logic based on resolution principle. An extended proof system described in section 5.3 is also essentially based on resolution principle. The procedure of the proof systems is to construct a sequence of clauses called deduction from a given set of clauses, and, in each step of extending the sequence, it makes resolution by unifying two formulae included in distinct clauses. The entire procedure stops when the expected clause is deduced[1].

Notice that there are two kind of nondeterminism in this procedure. One of them is a selection of a rule which expands the matching tree in a unification and the other is selections of clauses, literals and an mgu which are used in a resolution. On the other hand, in a proof system for

[1] In a refutation the expected clause is empty clause.

first-order predicate logic, the former nondeterminism does not exists and the later nondeterminism does not implies a selection of mgu because the number of mgu's of a set of first-order terms is at most one. Moreover, in a proof system for first-order predicate logic, two atomic formulae are unifiable only if both have a same constant symbol as their heads.

Therefore the proof system for higher-order clausal logic must be much less efficient than that for first-order predicate logic if we implement it in a straightforward way. To let the proof system some more efficient, we must consider some strategies to select clauses, literals and an mgu in a resolution or some restriction on the logic itself. However, in most cases, these expedients strongly depend on a purpose for which the proof system is used, therefore it is difficult to think out general strategies or restriction.

In this chapter, we discuss another some more general measure to implement the proof system efficiently, which is parallel implementation of the proof system. Although it does not improve the efficiency of the proof system so drastically, it improves steadily in general cases. In section 6.2, we make some discussion about parallelism in our proof system and we propose a processor network in section 6.3 which is adequate to the proof process.

6.2 Parallel Processing in the Proof Systems

The procedure of our proof systems can be separated in two levels. The lower one is a level of a resolution with unification. And the upper one is a level of extending a deduction sequence. As mentioned in the previous section, there is nondeterminism in each levels.

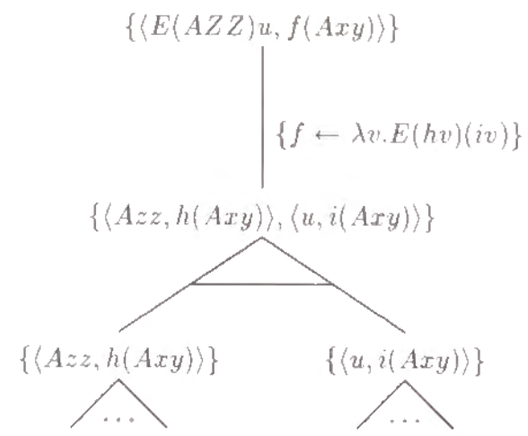


Figure 6.1: An example of AND-OR tree for unification

In the level of resolution, nondeterminism appears in unification procedure As described in chapter 2, the unification procedure is an expanding procedure for a matching tree, which is a kind of OR tree. This procedure can be easily modified, however, as an AND-OR tree expanding procedure by matching each pair in a disagreement set in a node separately though each subtree of AND branch are generally dependent because some variables may be shared in the each subtree(see figure 6.1).

In the level of deduction, there is similar structure if we consider a refutation only. A refutation can be regarded as an OR tree such that each node corresponds to goal or subgoal clause which should be refuted and each downward edges from the node corresponds to a clause to be resolved with the (sub)goal[2]. This OR tree can be also easily modified as

[2]Though it is arbitrary which clause should be a goal clause or a side clause, a clause corresponds to an OR edge, we can adopt a negative clause as a goal clause and other clauses as side clauses, if we consider Horn sets only.

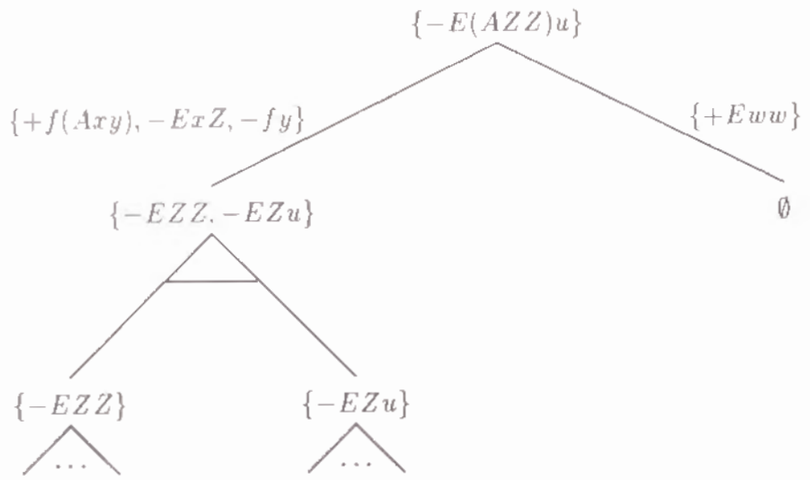


Figure 6.2: An example of AND-OR tree for refutation

an AND-OR tree if we consider that each literal in a goal clauses is refuted separately(see figure 6.2). In this level, each subtree of AND branch are also generally dependent because some variables may be shared in the each subtree.

In both levels, the AND-OR trees may be infinite. Undecidability in both levels is corresponding to the infinity. When we implement our deduction system, we must manage the infinity. For example, we cannot wait to finish an unification process in a step of deduction because it may takes infinite time. One way to manage the unfavorable situation is to combine both kinds of AND-OR trees. In such a combined AND-OR tree, each node has a subgoal and a disagreement set, a disagreement pair in which has appeared in a unification carried out to derive the subgoal and has not solved yet, and each edge corresponds to a side clause for a resolution or a expanding

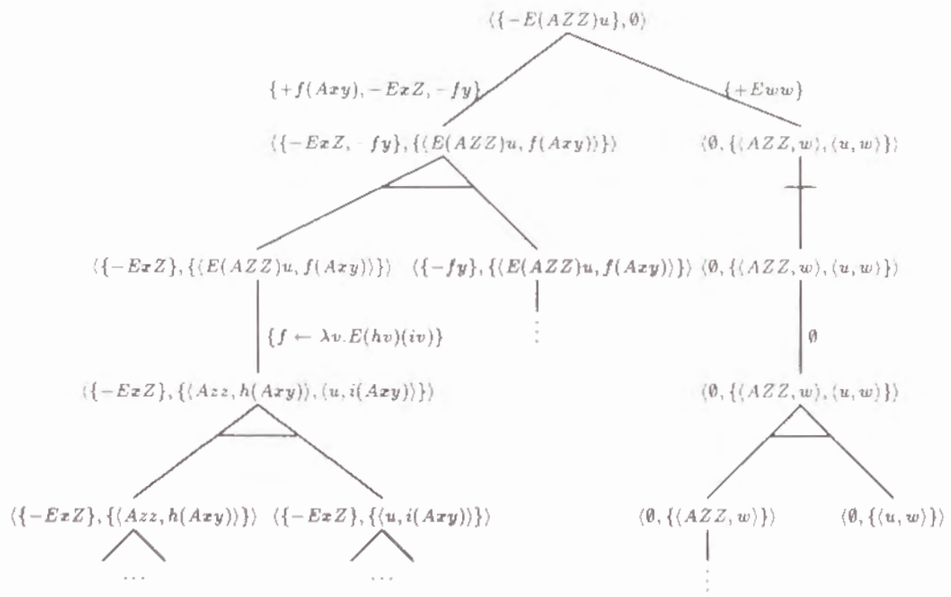


Figure 6.3: An example of combined AND-OR tree

rule to resolve disagreement set (i.e. a rule for matching tree)[3].(see figure 6.3)

Finally, we can regard the entire proof procedure as a procedure which expands an infinite AND-OR tree. When we consider a parallel implementation of our proof procedure, this observation of the procedure suggests two kinds of parallelism – AND parallelism and OR parallelism. Each of the parallelism means to let the subtrees under an AND/OR node be processed in parallel. When we implement a higher-order proof system such as ours, there are some reasons why the parallelism may be important. One

[3]This idea originated in the work of Miller and Nadathur[Miller 86], although their logic is different from ours and their proof procedure forces to execute a step for unification and a step for resolution one after the other.

reason is that because of the nondeterminism in the proof procedure, a sequential implementation must be too inefficient for practical use, therefore any kinds of technique for a good efficiency are needed. Another reason is that, because of the undecidability in the proof procedure, a considerable number of subtrees may be infinite, therefore a sequential process may have to do infinite search too often. [4].

If we can proceed subtrees under an AND node in parallel, it contributes the efficiency of the whole proof processes because each subtree should be proceeded anyway. However, as mentioned before, subtrees under an AND node are not independent because some variables are shared by the subtrees. Moreover, if a node becomes failed, the information must be sent to other processes which are processing other AND branches so that they can select another OR branch. Therefore we have to adopt some mechanism to let parallel processes communicate each other when we consider a parallel implementation of the proof system.

On the other hand, OR parallelism seems not to contribute the efficiency so much. However, to avoid infinite search, the parallelism is also effective. When we consider OR parallelism, there are similar problems to that in AND parallelism. In OR parallelism, each process corresponds to a branch of nondeterminism. Therefore it is needed to let the processes communicate each other for some kinds of global controls (such as to force terminating all processes).

In both cases – AND parallelism and OR parallelism, broadcasting seems to be a good communication method for global control such as selecting another OR branch or terminating all processes because these control should be done in all processes synchronously and triggered by an information such as "success" or "fail" in a process. For other communications, broadcasting is good enough because it implies point-to-point communication between any two processes.

[4]Of course there is no problem if we adopt breadth-first search. Breadth-first search itself, however, implies a kind of parallelism because it can be regarded as a sequential implementation of a parallel procedure.

Under the considerations, a processor network which is adequate to implement the proof procedure should be able to broadcast efficiently and reliably. In next section, we propose such a processor network and broadcasting scheme.

6.3 An Efficient Scheme for Broadcasting on a Processor Network

Broadcasting is a task initiated by a source processor that wishes to convey a message to all processors in a processor network. It can be accomplished by data dissemination in such a way that each processor sends a message to one of its neighbors at each round. This operation is very fundamental and important in a parallel and distributed computing system, especially in a parallel implementation of a proof system based on resolution principle as observed in last section.

Obviously, time required for broadcasting depends on topology of the network although, in general, time and the number of channels are related in trade-off.

Theoretical lower bound of broadcasting time on a network with N processors is $\lceil \log_2 N \rceil$ rounds. Hypercubes and binary jumping networks are well-known network models on which there are time optimal procedures

for broadcasting[Han 88][Igarashi 90][Johnsson 89] although the number of processors in the former model must be a power of two in contrast with the later in which the number of processors may be arbitrary. The numbers of channels in both of these network models is $O(N \log N)$ for the number of processors N.

The network model we propose in this section also has $O(N \log N)$ channels and is time optimal. However, there are some difference among these network models in their fault tolerance. Fault tolerance is a one of significant problems on reliability of a processor network. It is preferable that a reliable broadcasting can be efficiently completed even in a faulty network. When there are some faulty processors in each of the network models, required times for broadcasting in the network models are not necessarily same[Han 88][Liestman 85][Ramanathan 88]. Under the hypothesis that the number of faulty processor is at most one and that the faulty processor does not send any messages, it requires at least $\lceil \log_2(N-1) \rceil + 1$ rounds in worst case on any networks. On hypercubes and binary jumping networks, some procedures which takes $\lceil \log_2 N \rceil + 2$ rounds for broadcasting in worst case under the hypothesis are known[Han 88]. However, the time $\lceil \log_2 N \rceil + 2$ rounds is still larger than the lower bound. The network model we propose in this section is time optimal under the hypothesis except some special cases.

In subsection 6.3.1, we introduce the model of the processor networks and the scheme for broadcasting on it. We also prove that the scheme is optimal in efficiency. Subsection 6.3.2 are concerning to fault tolerance of the scheme. In the subsection, we show that the scheme is nearly optimal even though there is one faulty processor in the network.

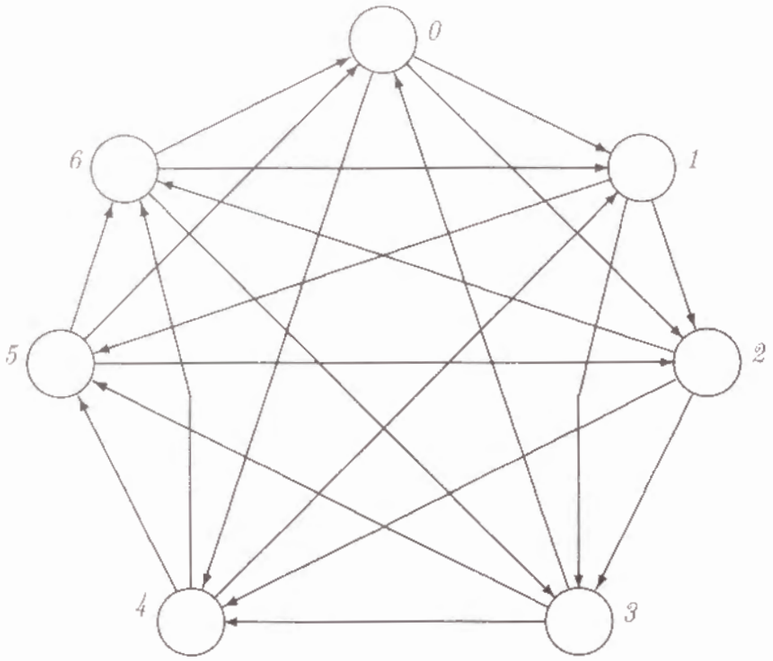
6.3.1 Information Dissemination Scheme

Concerning to a faulty processor network, there are various models corresponding to various conditions about faulty processor networks. We adopt the following conditions on our model of processor networks:

- All processors in a network work synchronously. The time unit synchronized with each other is called a round.
- Each channels are unidirectional. Two opposite channels are needed for bidirectional communication.
- Each processor in the network is able to send a message to only one processor per one round and then receive any numbers of messages through distinct channels.
- A faulty processor cannot send any messages although it is able to receive messages.

Under the conditions, the topology of our processor networks are defined as the following:

- Let N be the number of processors.
- Each processor is numbered uniquely with $0 \cdots N-1$.
- Let $n = \lceil \log_2 N \rceil$.
- Define $T[r]$ $(-1 \leq r \leq n-1)$ as the following recursively:
  - Let $T[-1] = N$.
  - Let $T[r] = \lceil T[r-1] \rceil$ $(0 \leq r \leq n-1)$



r	(-1)	0	1	2
$T[r]$	(7)	4	2	1

Figure 6.4: An example of our processor network ($N = 7$)

- There is a channel from a processor u to a processor v if and only if $(v - u) \bmod N = T[r]$ for a $T[r]$ $(0 \leq r \leq n - 1)$.

Example 6.1 Figure 6.4 shows the topology of the network and array T for $N = 7$. □

Obviously, a network with arbitrary number of processors can be defined by the definition and the number of channels of N processors network is Nn $(= N\lceil \log_2 N \rceil)$. Note that this network is symmetric for each processor. This fact reduces some proofs later.

The information dissemination procedure for broadcasting proposed in this thesis is as the following:

```
procedure disseminate(N,i)
  { dissemination procedure for i-th processor.
    N: the number of processors. }
  T: array of integer
begin
  n := ⌈log N⌉
  d := N
  for r := 0 to n − 1 do
    d := ⌈d/2⌉
    T[r] := d
  endfor
  while true do { outer loop (infinite loop) }
    for r := 0 to n − 1 do
      { send a message to processor (i + T[r]) mod N. }
      send((i + T[r]) mod N);
    endfor { inner }
  endwhile { outer }
end
```

This procedure is assumed to be executed by each of all processors in the network synchronously. First half of the procedure calculates value of n and values of $T[r]$. The rest of the procedure is for information dissemination. Since each iteration of inner loop corresponds to a round, we call each round "round r" with the number r for convenience. And so rounds of each processor circulates from 0 to $n - 1$ synchronously and, in a round r,

each processor in the network sends a message to the $T[r]$-th succeeding processor.

Example 6.2 Table 6.1 shows disseminating table for the network of example 6.1. Assume that the source processor is 2 and that the start round is 1. Table 6.2 shows the process of broadcasting. Obviously, broadcasting should be completed in three rounds. □

In example 6.2, broadcasting has been completed in n $(= \lceil \log_2 N \rceil)$ rounds. In fact, it can be proved that broadcasting should been completed in n $(= \lceil \log_2 N \rceil)$ rounds for any number N of processors, any start processor and any start round. Before we prove the fact, we prove a lemma.

Lemma 6.1 For any i such that $0 \leq i \leq n-1$, any j such that $0 \leq j < T[i-1]$ can be represented as

$$0[+T[i]][+T[i+1]]\cdots[+T[n-1]]$$

where each term formed as $[+\cdots]$ is an optional, which may be added or not.

Proof. We prove it with mathematical induction.

1. In the case where $i = n-1$, $j = 0$ or 1 because $0 \leq j < T[i-1] = T[n-2] = 2$. Hence j can be represented as $0[+T[n-1]](= 0[+1])$.

2. Otherwise, assume that the lemma holds in the case where $i = k+1$. Let j be an integer such that $0 \leq j < T[k-1]$.

   (a) In the case where $0 \leq j < T[k]$, j can be represented as $0[+T[k+1]]\cdots[+T[n-1]]$ because of assumption of induction. Then, j can be also represented as $0[+T[k]][+T[k+1]]\cdots[+T[n-1]]$ because it can be rewrite as $j = 0 + 0[+T[k+1]]\cdots[+T[n-1]]$.

Table 6.1: Dissemination table for $N = 7$

round		processors						
r	T[r]	0	1	2	3	4	5	6
0	4	4	5	6	0	1	2	3
1	2	2	3	4	5	6	0	1
2	1	1	2	3	4	5	6	0

Table 6.2: Broadcasting from processor 2 with start round 1

round		processors						
r	T[r]	0	1	2	3	4	5	6
1	2			o		o		
2	1			o	o	o	o	
0	4	o	o	o	o	o	o	o

o : informed processor at the end of the round

(b) In the case where $T[k] \leq j < T[k-1]$, $T[k-1] \leq 2T[k]$ holds immediately from the definition of T. Hence $j - T[k] < T[k-1] - T[k] \leq 2T[k] - T[k] = T[k]$ holds. From the assumption of induction with the inequality, $j - T[k]$ can be represented as $0[+T[k+1]] \cdots [+T[n-1]]$. Hence j can be represented as $0 + T[k][+T[k+1]] \cdots [+T[n-1]]$. This is implied in $0[+T[k]][+T[k+1]] \cdots [+T[n-1]]$.

From the results of the two cases, the lemma holds in the case where $i = k$.

Hence the lemma holds for any i such that $0 \leq i \leq n-1$ by mathematical induction. □

Now we can prove the optimality of the information disseminating scheme without any faulty processor.

Theorem 6.1 For any number N of processors, arbitrary start round r and arbitrary source processor s, broadcasting from s to all other processors is completed in $n = \lceil \log N \rceil$ rounds if no faulty processor exists.

Proof. Because of symmetry of the network, we may assume $s = 0$ without loss of generality. Then any processor which is informed in n rounds can be represented as $(0[+T[0]][+T[1]] \cdots [+T[n-1]])$ mod N for any start round r. On the other hand, from 6.1, any number j such that $0 \leq j < N\ (= T[-1])$ can be represented as $0[+T[0]][+T[1]] \cdots [+T[n-1]]$. Hence any processor j can be represented as $(0[+T[0]][+T[1]] \cdots [+T[n-1]])$ mod N, therefore the theorem holds. □

Remember that it is to only one processor that a processor can send a message per one round in our model. In such a model, informed processors

doubled per one round at most. Therefore it needs at least $\lceil \log N \rceil$ rounds to complete broadcasting in a network of N processors by any scheme for information dissemination. This means that our scheme is time optimal if there is no faulty processor.

6.3.2 Fault Tolerance

In this subsection, we consider the fault tolerance of our networks especially in the case where the number of faulty processor is at most one. As mentioned in the last subsection, we assume that a faulty processor does not send any messages. We give some definitions as preparations before presenting major result of this subsection.

Definition 6.1

$$\begin{cases} U_{-1} = \{0\} \\ U_i = U_{i-1} \cup \{u < N \mid u - T[i] \in U_{i-1} \text{ and} \\ \qquad\qquad \text{there is no } w \in U_{i-1} \text{ s.t. } u - T[i] < w \leq u\}, \\ \quad (0 \leq i \leq n-1) \end{cases}$$

□

If we assume that both start round and source processor are 0, each set U_i means a set of informed processors which receive broadcasted message until round i without any message sending which leaps over another informed processors. The next propositions are immediate.

Property 6.1 When we sort the elements in a U_i in ascending order then

1. the first two elements are 0 and $T[i]$,
2. differences between two adjacent elements are at least 1 and at most $T[i]$ and

3. the difference between the last element and N is also at least 1 and at most $T[i]$.

Proof. It is evident from definition 6.1. □

Definition 6.2 We define a binary relation $\prec$ on $\{0, 1, \cdots, N-1\}$ as the following:

$$u - T[i] \prec u \text{ iff } u \in U_i \text{ and } u \notin U_{i-1}.$$

We also denote the transitive closure of the relation as $\overset{+}{\prec}$. □

This relation means message passing between the processors in constructing U_i's. The next propositions are immediate.

Property 6.2 $u < v$ if $u \overset{+}{\prec} v$. And $u \in U_{i-1}$ if $u \overset{+}{\prec} v$ and $v \in U_i$ $(0 \leq i \leq N-1)$.

Proof. It is evident from definition 6.1. □

We classify the processors in U_i to two classes as a matter of convenience for proving the following lemma.

Definition 6.3 For $u \in U_i$, u called an intermediate processor in U_i iff there exists a processor $v \in U_i$ such that $u \prec v$, otherwise u called a terminal processor in U_i. □

Now, we prove a lemma about a terminal processor in U_i.

Lemma 6.2 Let u, v be elements of U_i $(0 \leq i \leq N-1)$ and $u < v$. Then u is a terminal processor if $v - u < T[i]$ holds.

Proof. At first, we assume that u which satisfies the condition of the theorem is an intermediate processor and then we show that the assumption implies a contradiction.

If u which satisfies the condition of the theorem is an intermediate processor, there exists a processor w such that $u \prec w$, $w - u = T[j] \geq T[i]$, $u \in U_{j-1}$ and $w \in U_j \setminus U_{j-1}$ for a j such that $j \leq i$.

On the other hand, because $u < v \neq 0$ and $v \in U_i$, there exists a processor x such that $x \prec v$, $v - x = T[k] \geq T[i]$, $x \in U_{k-1}$, and $v \in U_k \setminus U_{k-1}$ for a k such that $k \leq i$.

Hence, from $v - u < T[i]$, $x < u < v < w$.

1. In the case where $j \leq k$, $x(= v - T[k]) \in U_{k-1}$, $u \in U_{j-1} \subseteq U_{k-1}$, $u > x$ and $v > u$. Hence, from the definition of U_k, $v \notin U_k$ holds. It contradict the fact $v \in U_k \setminus U_{k-1}$.

2. In the case where $j > k$, $u(= w - T[j]) \in U_{j-1}$, $v \in U_k \subseteq U_{j-1}$, $v > u$ and $w > v$. Hence, from the definition of U_j, $w \notin U_j$ holds. It contradict the fact $w \in U_j \setminus U_{j-1}$.

Hence the assumption is false. That means that the theorem holds. □

Using this lemma, we get the next theorem about fault tolerance of our networks.

Theorem 6.2 Let N be the number of processors, r be a start round and s be a source processor which is not faulty. Then broadcasting from s completes at most $n+1$ $(= \lceil \log N \rceil + 1)$ rounds for any N, r and s if faulty processor is at most one.

Proof. The case where there is no faulty processor is implied in theorem 6.1. Hence we prove the case where the number of faulty processor is one. Because of the symmetry of the network, we can assume $s = 0$ without losing generality. Let f $(\neq 0)$ be the faulty processor. We divide our proof into three cases by the value of f.

1. In the case where $1 \leq f \leq T[r]-1$, let S_0 be a subset of $\{0 \cdots \min(2T[r]-1, N-1)\}$ in which each elements is not faulty processors and has received the broadcasted information certainly until the end of $(n-r)$-th round (round $n-1$), and let F_0 be the subset of $\{0 \cdots \min(2T[r]-1, N-1)\}$ in which each elements may not have received the information until the end of the round. Then

$$S_0 = \{g \bmod N | g = 0 \text{ or } T[r] \leq g \leq 2T[r]-1\}$$

and

$$F_0 = \{g | 1 \leq g \leq T[r]-1\}$$

because $2T[r]-1 \leq N$.

Hence, after the dissemination through to n-th round (round $r-1$), the set of processors which has received the information certainly and the set of processors which may not have received the information are

$$S = \bigcup_{u \in U_{r-1}} (u + S_0)$$

and

$$F = \bigcup_{u \in U_{r-1}} (u + F_0)$$

respectively[5].

Then, the set of processors, each of which is in a position $T[r]$-th forward from a processor in F, is

$$\begin{aligned} F' &= \bigcup_{u \in U_{r-1}} ((u + T[r]) + F_0) \\ &= \bigcup_{u \in U_{r-1}} (u + \{(g + T[r]) \bmod N | 1 \leq g \leq T[r]-1\}) \end{aligned}$$

[5] $u + \{g_1, g_2, \cdots, g_k\}$ means $\{(u+g_1) \bmod N, (u+g_2) \bmod N, \cdots, (u+g_k) \bmod N\}$.

$$= \bigcup_{u \in U_{r-1}} (u + \{g' \bmod N | T[r]+1 \leq g' \leq 2T[r]-1\})$$

Therefore it is included in S.

On the other hand, let v be a processor which has not receive the information until round $r-1$. Then $v \notin S$ holds. Hence, by contraposition of the former result, $w \notin F$ holds where $w = (v - T[r]) \bmod N$. Additionally w is not faulty because$(f + T[r]) \bmod N \in S_0 (\subset S)$.

Hence v can receive the information from w in the $n+1$-th round (second round r).

2. In the case where $T[r] \leq f \leq T[r-1]-1$, let S_0 be a set of processors in which each elements is not faulty and has received the broadcasted information certainly until the end of $(n-r)$-th round (round $n-1$) and let F_0 be the set of processors in which each elements may not have received the information until the end of the round. Then

$$S_0 = \{g | 0 \leq g \leq T[r]-1\}$$

and

$$F_0 = \{g | T[r] \leq g \leq T[r-1]-1\}$$

Hence, after the dissemination through to n-th round (round $r-1$), the set of processors which has received the information certainly and not faulty and the set of processors which may not have received the information are

$$S = \bigcup_{u \in U_{r-1}} (u + S_0)$$

and

$$F = \bigcup_{u \in U_{r-1}} (u + F_0)$$

respectively.

Then, the set of processors, each of which is in a position $T[r]$-th backward from a processor in F, is

$$\begin{aligned} F' &= \bigcup_{u \in U_{r-1}} ((u - T[r]) + F_0) \\ &= \bigcup_{u \in U_{r-1}} (u + \{(g - T[r]) \bmod N \mid T[r] \leq g \leq T[r-1] - 1\}) \\ &= \bigcup_{u \in U_{r-1}} (u + \{g' \bmod N \mid 0 \leq g' \leq T[r-1] - T[r] - 1 \leq T[r] - 1\}) \end{aligned}$$

Therefore it is included in S.

Hence any processor in F can receive the information in the $n+1$-th round (second round r).

3. In the case where $T[r-1] \leq f \leq N-1$, each processor in $\{g | 0 \leq g \leq 2T[r]-1\}$ has received the broadcasted information until the end of $(n-r)$-th round (round $n-1$). Let $u_0, u_1, \cdots, u_{n_r-1}$ be the ascending sequence of all elements in U_{r-1}. Then there exists an integer i such that $i > 1$ and $u_i \leq f \leq u_{i+1} - 1$[6]. Let U' be the subset of U_{r-1} such that $U' = \{u | u \in U_{r-1}, (0 \leq) u_i - T[r-1] < u \leq u_i\}$. Then apparently $u_i \in U'$ holds. On the other hand, any element in U' is a terminal processor in U_{r-1} except u_i because of lemma 6.2. Hence, for any intermediate processor u in U_{r-1}, $f < u$ or $u + T[r-1] \leq f$ if $u \neq u_i$. We divide the proof into two subcases again.

[6] If $u_{n_r-1} \leq f (\leq N-1)$, then let $i = n_r - 1$.

(a) In the case where u_i is a terminal processor, arbitrary processor v can be represented as $u_j + t$ where $u_j \in U_{r-1}$ and $0 \leq t < T[r-1]$. Considering the information passing path from 0 to v, the information reaches processor t until the end of $(n-r)$-th round (round $n-1$). And then, through to n-th round (round $r-1$), the information passes from t to v via $v_1+t, v_2+t, \cdots, v_{k-1}$ where $v_i \in U_{r-1}$, $0 \prec v_1 \prec \cdots \prec v_k$ and $v_k = u_j$. Hence at least n rounds suffice for the broadcasting.

(b) In the case where u_i is a intermediate processor, let F be the set of processors which may not have received the information until the end of $(n-r)$-th round (round $r-1$). Then

$$F = \bigcup_{u \in U_{r-1}, u_i \overset{+}{\prec} u} \{(u + (f - u_i)) \bmod N\}$$

We divide the proof into three subsubcases furthermore.

i. In case where $u_i \leq f \leq u_i + T[r] - 2$, the set of processors each of which is in a position $T[r]$-th forward from a processor in F, is

$$\bigcup_{u \in U_{r-1}, u_i \overset{+}{\prec} u} \{(u + (f - u_i + T[r])) \bmod N\}$$

Then, because $T[r] \leq f - u_i + T[r] \leq 2T[r] - 2 \leq T[r-1] - 1$, each processor in the set has received the information via $f - u_i + T[r]$. It means that any processors which have not received the information (until the end of $(n-r)$-th round) never be included in this set. Hence processors in the position $T[r]$-th backward from the (not informed) processors are not included in F. Therefore broadcasting completes in the $(n+1)$-th round (second round r).

ii. In the case where $f = u_i + T[r] - 1$, If we assume $u + 2T[r] - 1 = f$ for a $u \in U_{r-1}$, u must be a terminal processor because $u_i - u = T[r] < T[r-1]$ and lemma 6.2. Hence, for any intermediate processor u, $f < u$ or $u + 2T[r] \leq f$ if $u \neq u_i$. Because $f - u_i + T[r] = 2T[r] - 1$, broadcasting completes in $n + 1$ rounds as in the case 3(b)i.

iii. In the case where $u_i + T[r] \leq f \leq u_{i+1} - 1 \leq u_i + T[r-1] - 1$ (or $u_i + T[r] \leq f \leq N - 1 \leq u_i + T[r-1] - 1$ if $i = n_r - 1$), the set of processors each of which is in a position $T[r]$-th backward from a processor in F, is

$$\bigcup_{u \in U_{r-1}, u_i \overset{+}{\prec} u} \{(u + (f - u_i - T[r])) \bmod N\}$$

Then, because $0 \leq f - u_i - T[r] \leq T[r-1] - T[r] - 1 \leq T[r] - 1$, each processor in the set has received the information via $f - u_i + T[r]$. Therefore broadcasting completes in the $(n+1)$-th round (second round r).

Consequently, the theorem has proved in any case. □

In the case where there is one faulty processor, in worst case, it needs at least $\lceil \log(N-1) \rceil + 1$ rounds to complete broadcasting by any disseminating scheme[7]. This lower bound is equal to the upper bound of our scheme except the case where $N = 2^m + 1$ (m is an arbitrary integer).

[7]Consider the case where the faulty processor is the processor to which the source processor sends a message in the start round.

6.4 Remarks and Discussions

In this chapter, we proposed a synchronous processor network and an information disseminating scheme for broadcasting on the network. The network and the information disseminating scheme have the following features:

- Any number of processors can be possible in contrast with, for example, hypercubes.
- The number of channels in the network is $N\lceil \log_2 N \rceil$ where N is the number of processors.
- The network is symmetric for each processor. Therefore the information disseminating scheme works in similar way for any source processor.
- The information disseminating scheme displays its best performance in broadcasting from any start round.
- The information disseminating scheme is time optimal if no processor is faulty.
- The information disseminating scheme is time optimal even if a single processor is faulty except the number of processors is a power of two. In the case where the number of processors is a power of two, required time for broadcasting exceeds the optimal time by at most one.

In the case where the number of processors is a power of two, it is unknown that there exists an information disseminating schema which can always complete the broadcasting in $\log_2 N$ rounds even if a single processor is faulty. Therefore our schema may be optimal as a matter of fact.

This scheme is easily extended such that each processor is able to send multiple messages to distinct processors per one round. However, it is unknown whether the extended scheme is better or not than other schemes [Kanai 90].

On the other hand, there is some models of networks in which the number of channels are $O(N)$ and on which the time required for broadcasting is $O(\log N)$[Bermond 88][Imase 85]. However, it is not time optimal and no procedure is known which can broadcast from any processor in any start round with constant efficiency. Moreover, fault tolerance of the models are unknown.

As mentioned in section 6.2, broadcasting performs an important role in a parallel implementation of our proof system. Therefore the processor network and the information disseminating scheme which we proposed in section 6.3 should be a good basis for such an implementation.

To implement our proof system on the network concretely, there are still more problems. In a parallel implementation on our proof system, a lot of processes should be generated and terminated dynamically. Therefore it is a significant problem how to assign and schedule the processes on the concrete processors in the network. It is also a problem how to control the AND/OR tree expanding globally. However these problems seem to depend strongly on a specific implementation and so they are beyond this thesis.

Chapter 7

Concluding Remarks

In chapter 2, we proposed higher-order clausal logic. This logic is an extension of first-order predicate logic with Skolem function, and has more flexibility and ability to describe various matters as shown in examples in chapter 4 and chapter 5.

In chapter 4, we defined the class of H-model as an extension of Herbrand model on a type restricted higher-order clausal logic. And then we showed that a proof system based on resolution principle is complete for the model class. Although the logic system is syntactically restricted, any term of first-order predicate logic is allowed. Moreover, a term of any type which is constructed solely from 1 is also allowed. Therefore the logic system is properly extension of first-order predicate logic. On the other hand, as mentioned at the last of the chapter, the syntactic restriction may not be unique. It is a future works to find other restrictions which is better in a sense.

In chapter 5, we defined the class of denotational model without any type restriction dissimilarly to H-model. And then we showed that a proof

system based on resolution principle with equality axioms is complete for the model class. However, it is not clear that this extension is redundant or not, or there may be any other extension, for example, including not axioms but inference rules. These problems are left for future works.

Both of the two model classes are extension of Herbrand model in first-order predicate logic. And so they are strongly related each other. Figure 5.2 shows the relations between standard model, generalized model, H-model (Herbrand model) and denotational model.

Both in the model classes, H-model and denotational model, any element in a domain of a model has at least one corresponding expression in H-universes. The feature is adequate when we consider higher-order clausal logic as a logic programming system because it seems to be appropriate to exclude the element which has no corresponding expression. In fact, when we consider such a logic programming system, the completeness we proved for the model classes means that, for any program, there exists a finite execution of the program if the program with input data has a valid symbolic solution. Essentially, it is impossible to get any nonsymbolic solution, and so the domain restriction in H-model or denotational model should not be disadvantage.

Concerning to a concrete implementation, we proposed a model of processor network and an information disseminating scheme in chapter 6. The model of processor network could be composed with any number of processors and the information disseminating scheme is optimal for broadcasting in most cases if the number of faulty processors is at most one. For more faulty processors, it is unknown that the scheme is more efficient than other scheme. To find an upper bound of the time for broadcasting in the case where the number of faulty processors is more than one is a problem left

for future works.

The information disseminating scheme could be easily extended to send messages to more than one distinct processors per one rounds. The behavior in the case is another problem for future works.

For practical application of higher-order clausal logic, efficiency problem is still essential. To advance the efficiency still more, application specific technique should be needed. One of expected application is an amalgamation of logic programming and functional programming[Miura 88a] [Miura 88b]. In the case, both programming style should be treated in a single semantic framework. However, executions in the system may be modified for its efficiency within the limits in which the results are compatible with the semantics. Especially, the part of functional programming system might be optimized as if it is purely functional system.

Program verification and program generation are also expected application. However, much more researches should be needed for its implementation.

Acknowledgments

The author would like to express sincere appreciation to Professor Shuji Doshita of Kyoto University for the supervision and encouragement to complete this thesis. He also present the author proper guidance which greatly contributed to the improvement of the thesis.

The author would like to express sincere thanks to Dr. Susumu Yamasaki for helpful discussions and accurate comments. He also guided the author to the present study.

The author wish to thank Professor Yoshihide Igarashi of Gunma University for the proper guidance and advice especially on parallel distributed processing.

The author wish to thank and regret Mr. Youichi Shimada sincerely. His accurate comments and discussions were very helpful to formulate the thesis. His premature death was seriously heavy loss for us. The author also wish to thank other members of Prof. Doshita's laboratory who have taken a time for discussions and provided many helpful comments.

The author's debt is to the member of Prof. Igarashi's laboratory for helpful discussions and cordial supports. Especially, Mr. Shingo Osawa and Mr. Kouji Obokata helped the author with computer simulation of parallel processing.

Finally, the author wish to thank especially Professor Toyoaki Nishida of Advanced Institute of Science and Technology, Nara for his supports and encouragements.

References

[Bermond 88]

J-C. Bermond and C. Peyrat: "Broadcasting in de Bruijn networks", *Congressus Numerantium*, Vol. 66, pp. 283-292 (1988)

[Bienstock 88]

D. Bienstock: "Broadcasting with random graphs", *Discrete Appl. Math.*, Vol. 20, pp. 1-7 (1988).

[Chang 73]

C. L. Chang and R. C. T. Lee: "Symbolic logic and mechanical theorem proving", Academic Press (1973).

[Feige 90]

U. Feige, D. Peleg and P. Raghavan: "Randomized broadcast in networks", *Proc. of SIGAL International Symposium on Algorithms*, Tokyo, Japan, *LNCS*, Vol. 450, pp. 128-137 (1990).

[Fraigniaud 89]

P. Fraigniaud: "Performance analysis of broadcasting", *Proc. of 1st European Workshop on Hypercubes and Distributed Computers*, Rennes, France, pp. 331-327 (1989).

[Gödel 31]

K. Gödel: "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme", *Monatshefte Math. Phys.*, Vol. 38, pp. 173-198 (1931).

[Goldfarb 81]

W. D. Goldfarb: "The undecidability of the second-order unification problem", *Theoretical Computer Science*, Vol. 13, pp. 225-230 (1981).

[Han 88]

Y. Han and R. Finkel: "An optimal scheme for disseminating information", *Proc. of 1988 International Conference on Parallel Processing*, Chicago, Illinois, pp. 198-203 (1988).

[Hedetniemi 88]

S. M. Hedetniemi, S. T. Hedetniemi and A. L. Liestman: "A survey of gossiping and broadcasting in communication networks", *Networks*, Vol. 18, pp. 319-349 (1988).

[Henkin 50]

L. Henkin: "Completeness in the theory of types", *J. of Symbolic Logic*, Vol. 15, pp. 81-91 (1950).

[Huet 75]

G. P. Huet: "A unification algorithm for typed λ-calculus", *Theoretical Computer Science*, Vol. 1, pp. 27-57 (1975).

[Igarashi 90]

Y. Igarashi, K. Kanai and K. Miura: "Information disseminating schemes

for fault tolerance in hypercubes", *Technical Report of IEICE*, COMP90-28, pp. 81-86 (1990).

[Imase 85]

M. Imase, T. Soneoka and K. Okada: "Connectivity of regular directed graphs with small diameters", *IEEE Trans. Comput.*, Vol. 34, pp. 267-273 (1985).

[Johnsson 89]

S. L. Johnsson and C.-T. Ho: "Optimal broadcasting and personalized communication in hypercubes", *IEEE Trans. Comput.*, Vol. 38, pp. 1249-1268 (1989).

[Kanai 90]

K. Kanai, K. Miura and Y. Igarashi: "Information disseminating schemes for fault tolerance in computer networks", *Technical Report of IEICE*, COMP90-35, pp. 45-52 (1990).

[Kowalski 74]

R. A. Kowalski: "Predicate logic as programming language", *Proc. of IFIP-74 Congress*, North-Holland, 1974, pp. 569-574.

[Liestman 85]

A. L. Liestman: "Fault-tolerant broadcast graphs", *Networks*, Vol. 15, pp. 159-171 (1985).

[Miller 86]

D. A. Miller and G. Nadathur: "Higher-order logic programming", *Proc. of 3rd international conference on logic programming, LNCS*, Vol. 225, pp. 448-462, Springer-Verlag, (1986).

[Miura 88a]

K. Miura and S. Doshita: "Amalgamation of term rewriting system and logic programming on higher-order clausal logic", *Technical Report of the Professional Group on Knowledge Engineering and Artificial Intelligence of IPSJ*, 58-4, (1988).

[Miura 88b]

K. Miura and S. Doshita: "Amalgamation of term rewriting system and logic programming on higher-order clausal logic", *Technical Report of JSAI*, SIG-FAI-8801-4, (1988).

[Pietrzykowski 72]

T. Pietrzykowski and D. C. Jensen: "A complete mechanization of (ω)-order type theory", *ACM national conf.*, Vol. 1, pp. 82-92 (1972).

[Ramanathan 88]

P. Ramanathan and K. G. Shin: "Reliable broadcasting in hypercube multiprocessors", *IEEE Trans. Comput.*, Vol. 37, pp. 1654-1657 (1988).

[Robinson 65]

J. A. Robinson: "A machine oriented logic based on the resolution principle", *J. of ACM*, Vol. 12, 1, pp. 23-41 (1965).

List of Publications

Major Publications

1. K. Miura, S. Yamasaki and S. Doshita: "On Herbrand model for higher-order clausal logic", *IEICE Trans.*, Vol. J72-D-I, No. 2, pp. 83-91 (1989), (In Japanese).

2. K. Miura, S. Doshita and S. Yamasaki: "On a proof system with identity axioms in higher-order clausal logic", *IEICE Trans.*, Vol. J72-D-I, No. 12, pp. 845-855 (1989), (In Japanese).

3. K. Miura, S. Osawa, Y. Igarashi and K. Kanai: "An efficient scheme for disseminating information on a processor network including at most one faulty processor", *IEICE Trans.*, Vol. J74-D-I, No. 12, pp. 869-875 (1991), (In Japanese).

4. Y. Igarashi, K. Kanai, K. Miura and S. Osawa: "Optimal schemes for disseminating information and their fault tolerance", *IEICE Trans. Inf. and Syst.*, Vol. E75-D, No. 1, pp. 22-29 (1992).

5. S. Carlsson, Y. Igarashi, K. Kanai, A. Lingas, K. Miura and O. Petersson: "Information disseminating schemes for fault tolerance in hypercubes", *IEICE Trans. Fundamentals*, Vol. E75-A, No. 2, pp. 255-260

(1992).

6. K. Kanai, Y. Igarashi and K. Miura: "Fault tolerance of an information disseminating scheme on a processor network", *IEICE Trans. Fundamentals*, Vol. E75-A, No. 11, pp. 1555-1560 (1992).

7. Y. Han, Y. Igarashi, K. Kanai and K. Miura: "Fault-tolerant broadcasting in binary jumping networks", *Third International Symposium on Algorithms and Computation (ISAAC '92)*, Nagoya, Japan, *LNCS*, Vol. 650, pp. 145-154 (December 1992).

Other publications

1. K. Miura and S. Doshita: "On the completeness in higher-order logic with the resolution principle", *Technical Report of IECE*, AL85-4, pp. 27-34 (1985), (In Japanese).

2. K. Miura, S. Yamasaki and S. Doshita: "On Herbrand interpretation for higher-order clausal form logic", *Proc. of 1st Annual Convention JSAI*, 1-7, pp. 33-36 (1987), (In Japanese).

3. K. Miura, S. Yamasaki and S. Doshita: "On completeness in higher-order clausal form logic with equality", *Proc. of 35th Annual Convention of IPSJ*, 5B-4, pp. 71-72 (1987), (In Japanese).

4. K. Miura and S. Doshita: "Amalgamation of term rewriting system and logic programming on higher-order clausal logic", *Technical Report of the Professional Group on Knowledge Engineering and Artificial Intelligence of IPSJ*, 58-4, (1988), (In Japanese).

5. K. Miura and S. Doshita: "Amalgamation of term rewriting system and logic programming on higher-order clausal logic", *Technical Report of JSAI*, SIG-FAI-8801-4, (1988), (In Japanese).

6. Y. Igarashi, K. Kanai and K. Miura: "Information disseminating schemes for fault tolerance in hypercubes", *Technical Report of IEICE*, COMP90-28, pp. 81-86 (1990).

7. K. Kanai, K. Miura and Y. Igarashi: "Information disseminating schemes for fault tolerance in computer networks", *Technical Report of IEICE*, COMP90-35, pp. 45-52 (1990), (In Japanese).

8. Y. Igarashi, K. Kanai K. Miura and S. Osawa: "Optimal schemes for disseminating information and their fault tolerance", *Proc. Forth KARUIZAWA Workshop on Circuits and Systems*, pp. 87-91 (1991).

9. A. Mei, K. Kanai, S. Osawa, K. Miura and Y. Igarashi: "Exact evaluation for broadcasting time on a network with faulty processors", *Technical Report of IPSJ*, AL22-10 (1991).

10. K. Kanai, Y. Igarashi, and K. Miura: "Fault tolerance of Han-Finkel's scheme for disseminating information", *Technical Report of IPSJ*, AL24-2 (1991).

11. K. Kanai, Y. Igarashi, and K. Miura: "Fault tolerance of an information disseminating schemes on a processor network", *Technical Report of IEICE*, CPSY91-60, pp. 61-68 (1991).

12. Y. Han, Y. Igarashi, K. Kanai and K. Miura: "Broadcasting in faulty binary jumping networks", *Technical Report of IPSJ*, AL26-7, pp.49-56 (1991).

13. Y. Igarashi, K. Obokata and K. Miura: "On fault tolerance of binary jumping information dissemination", *Technical Report of IEICE*, COMP92-16, pp. 1-6 (1992), (In Japanese).

IECE = The Institute of Electronics and Communication Engineers of Japan

IEICE = The Institute of Electronics, Information and Communication Engineers of Japan

IPSJ = Information Processing Society of Japan

JSAI = Japaanese Society for Artificial Intelligence

Appendix A

Notations

A.1 Notations in Chapter 2, 3, 4, 5

$\emptyset$	empty set
$S \setminus A$	set subtraction of A from S
$S/\simeq$	quotient set of S by a binary relation $\simeq$
$0, 1$	type constants
s, t, q, r	metasymbols for types
$\langle st \rangle$	a type constructed from s and t
T	set of all types
C_t	a set of constants of type t
V_t	a set of variables of type t
$\mathcal{V}$	a vocabulary
$\mathcal{V}^+$	an extended vocabulary in chapter 5
$x, y, z, u, v, w, f, g, \cdots$	metasymbols for variables, or variables themselves

$A, B, F, G, P, Q, R, \cdots$	metasymbols for constants, or constants themselves
$\alpha, \beta, \gamma, \cdots$	metasymbols for terms or term classes (including atomic formulae)
α^t	a term of type t
$(\alpha\beta)$	a term constructed from term α and term β (function application)
$(Fx_1x_2\cdots x_m)$	abbreviation of $(\cdots((Fx_1)x_2)\cdots x_m)$
$(\lambda x.\alpha)$	a term constructed from variable x and term α (lambda abstraction)
$(\lambda x_1x_2\cdots x_n.\alpha)$	abbreviation of $(\lambda x_1.(\lambda x_2.(\cdots(\lambda x_n.\alpha)\cdots)))$
$\mathcal{T}$	a set of terms
$\mathcal{T}_\mathcal{V}$	set of all terms on vocabulary $\mathcal{V}$
$\overset{\alpha,\beta}{\equiv}$	an equivalence relation based on α- and β-conversion
$+\alpha, -\alpha$	a positive literal and a negative literal constructed from atomic formula α
$L_1, L_2, \cdots$	metasymbols for literals
$\mathcal{C}, \mathcal{C}', \mathcal{C}_1, \mathcal{C}_2, \cdots$	metasymbols for clauses or subset of clauses
$\mathcal{S}, \mathcal{S}', \cdots$	metasymbols for sets of clauses
$\{x_1 \leftarrow \alpha_1, \cdots, x_n \leftarrow \alpha_n\}$	a substitution
$\theta, \pi, \rho, \sigma, \cdots$	metasymbols for substitutions
D_t	a domain of values of type t
$[D_t]_{t\in T}$	a frame (including g-frame and so on)
m_t	meaning function for constants of type t

$[D_t, m_t]_{t\in T}$	a model (including g-model and model on a model class)
$M, M', \cdots$	metasymbols for models
a_t	assignment for variables of type t
$[\mathrm{a}_t]_{t\in T}$	an assignment for variables
$\mathbf{a}$	metasymbol for an assignment
$V_{M\mathbf{a}}$	value function on M with $\mathbf{a}$
$[H_t]_{t\in T}$	higher-order extension of Herbrand universe
H_0	Herbrand base
$I, I', \cdots$	H-interpretation or partial H-interpretation (a set of literals)
$[\Phi_t^M]_{t\in T}$	a mapping for higher-order Herbrand model
$[\Psi_t^I]_{t\in T}$	an isomorphism for denotational model
B	a semantic tree
$N, N', \cdots$	nodes of a semantic tree
$I(N)$	a partial H-interpretation corresponding to node N
$\sim$	a binary relation among terms (term classes)
$\equiv_t$	special constant symbols in equality axioms
$\Gamma_{s,t}$	special constant symbols in equality axioms
$\mathcal{E}$	set of equality axioms

A.2 Notations in Chapter 6

$\lceil x \rceil$	ceiling of x
N	number of processors

n	$= \lceil \log_2 N \rceil$
$T[r]$	send distances on round r
U_i	a set of processors defined in chapter 6
$\prec$	a binary relation among processors
$\overset{+}{\prec}$	transitive closure of $\prec$